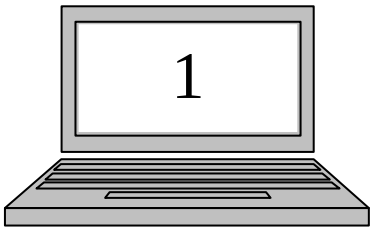


CHAPTER ONE

Basics

IN THIS CHAPTER

- Byte Code
- Features of Java
- JDK & its Components
- Whitespace in Java
- Identifiers in Java
- Comment in Java
- Keywords in Java
- Literals in Java
- Compiling and Running Java Programs
- Understanding Basic Java Program
- Primitive Data-Types Of Java
- Arrays
- Find Length Of Array
- Operators
- Type Conversion and Casting
- Garbage Collection
- Finalize()Method
- Conditional Statements
- Looping Statement
- Branching and Jumping Statement

1) History of java

It is related to c++ which is direct decedent of c much of character of java is inherited from this 2 language, form c java drives it syntax. Many of java's object oriented features where influents by c++.

Java was introduced by jems gosling, patric naughton, chris warthm ed frank & mike Sheridan at Sun Microsystems in 1991.

It took 18 months to develop the 1st version. This language was initially called OAK. But it was renamed java in 1995.

2) Byte code

The key that allows to java to solve both the security & portability problems are that the out of java compiler is not executable code. It is known as bytecode.

It is a highly optimize set of instruction design to be executed by the java runtime system which is called JVM (Java Virtual Machine). The original JVM was designed as an interpreter for byte code

Translating java program into byte code makes it much easier to run a program in a wild variety of environments because only the JVM needs to be implemented for each platform.

The fact that java program is executed by JVM also help to make it secure.

Because the JVM in control it can contain the program & prevent it from generating side effects outside of system, Sun begin supplying its hotspot technology that provides a JIT *(Just In Time) compiler for byte code

When a JIT compiler is a part of JVM selected part of byte code are compile into executable code in real time on peas by peas demand based

It is important to understand that it will not compile an entire java program in to executable code all at once.

3) Features of Java

Java has following features.

Simple
Similar to C++ but yet relieve the complexity and unnecessary features of c++ like structure, union, pointers. Automatic system for allocation and de-allocation of memory.

Object Oriented
Object is a combination of data and operations that can be performed on that data called method. Classes in the java are the example of Object Oriented nature of java programming language Java also supports the mechanism of Abstraction, encapsulation, inheritance etc.

Robust
Java does not crash easily because of programming bugs or error that is why it is called robust programming language.

Dynamic
In C++ change in one module needs to re-compile the whole code where in java it allows to change or add another module without affecting their dependent client objects.

Secure
Java does not allow the pointers so that one cannot access the restricted memory location by any means in java. For data java uses public key encryption technique to authenticate the data.

Architecture Neutral
Byte code generated by the java compiler can be run on any other operating system having java virtual machine installed. So the architecture of OS is not restriction for java language.

Portable
In C or C++ code may run slightly differently on different hardware because it depends on how the each hardware interprets the arithmetic calculation. Here in java an integer is always signed 32 bit integer and real value, float, is always 32 bit real value so this consistency produces the same result on any computer and hence it is called as portable language.

Distributed
Unlike C and C++ java is designed to work on network area. Large library of java classes provides mechanism to access the protocol suites like HTTP, TCP/IP and FTP. By this a programmer can handle any remote data or resources same as handling a local data or resources.

Multithreaded
Java supports the feature of multi threading where an application can run more then one thread at a time.

Interpreted
Java Source file --> Java compiler --> Byte code --> Java interpreter So any machine that runs java interpreter can run this code.

4) JDK & its Component

JDK: Java Development Kit provides the collection of tools that are used to develop and run Java programs. The tools that JDK provides are as follows:

appletviewer (for Java applets): Runs the applet application.
javac (Java compiler): Translates the java source code to the byte code.
java (java interpreter): Runs the application by interpreting bytecode.
javap (Java disassembler): Helps to convert the byte code file into a program description
javah: for C header files
javadoc: HTML documents having information about the java source code.
Jdb (Java debugger): To help the programmers in finding errors and debugging the current programs.

5) Whitespace in Java

Java is a free form language it means one can write code the way he/she wants. There is no such restriction like next line of code should come immediate after one line.

6) Identifiers in Java

Identifiers normally known as variables. Variables can have any alphabets or alphabets in combination with number and special chars like '_' and '$'. Variables cannot start from digits. Since java is case sensitive language "var ABC" and "var abc" are two different variables. There cannot be a white space between variables.

For Example:

String $Demo, _Demo, Demo1, De1mo; all are correct variable declaration.
String 1Demo, *Demo, 132Demo; all are incorrect variable declaration.

7) Comment in Java

Two types of comments in java.

- Single Line Comment: uses '//' syntax to comment the line.
- Multi Line Comment: uses '/*…….*/'

8) Keywords in Java

The keywords means reserved word or predefine word of Java.

abstract	continue	goto	package	synchronized
assert	default	if	private	this
void	boolean	do	implements	protected
throw	break	double	import	public
throws	volatile	byte	else	instanceof
return	transient	case	extentds	int
short	try	catch	final	interface
static	char	finally	long	strictfp
classes	float	native	super	while
const	for	new	switch	

9) Literals in Java

a) Character Literals

\n	New line
\t	Tab
\b	Backspace
\r	Carriage return
\f	Formfeed
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\d	Octal
\xd	Hexadecimal

b) Integer Literals:

- Decimal integer literal of type long integer
- Hexadecimal integer literal of type integer
- Octal integer literal of type long integer

c) Boolean Literals:

- true
- false

d) Floating point Literals:

12.8098

e) String Literals:

```
“”                      //the empty string
”\””                    //a string containing “
”This is a string”      //a string containing 16 characters
”This is a “ +          // actually a string-valued constant expression,
”two-line string”       // formed from two string literals
```

Strings can include the character escape codes as well, as shown here:

```
String example = “Your Name, \”abc\””;
System.out.println(“Thankingyou,\nabc\n”);
```

10) Compiling and Running Java programs

First create a program and save it. To save the program give the name of class which contains main() method. Because every console base java program execution start with the class which contains main() method.

It is not compulsory to give program name same as class name which contains main() method. But it is compulsory to execute or run that program with the class name which contains main() method.

For example

```
class abc
{
        public static void main(String s[])
        {
                ……….;
        }
}
```

Save the above program with the name of **abc.java** or any other name like **xyz.java**
Compile the above program are as follows

javac abc.java **or** **javac xyz.java**

Run / Execute the program are as follows.

java abc

11) Understanding basic java program

```
class abc
{
        public static void main(String s[])
        {
                System.out.println(" Hello ");
        }
}
```

class

Keyword to create class.

abc

Name of class (identifier).

{ }

The entire class definition, including its entire member (variable & methods) will be between the opening curly brace ({) and the closing curly brace (}).

public

- The public keyword is an access specifier, which allows the programmer to controlthe visibility of class members. When a class member is preceded by public, then thatmember may be accessed by code outside the class in which it is declared.
- In this case, **main( )** must be declared as **public**, since it must be called by code outside of its class when the program is started.

static

The keyword static allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since main( ) is called by the Java interpreter before any objects are made.

void

Keyword which indicates that main() method does not any value.

main()

Method at which program execution is started.

String s[]

String s[ ] declares a parameter named s, which is an array of instances of the class String. (*Arrays* are collections of similar objects.)

System.out.println()

System is a predefined class that provides access to the system, and **out** is the output stream that is connected to the console. **println()** is the method to print String type value on console.

" Hello "

String literal to print on console.

12) Primitive Data-types of Java

a) byte

- Size : 1 byte
- Default Value : 0

b) short

- Size : 1 byte
- Default value : 0

c) long

- Size : 8 byte
- Default value : 0

d) int

- Size : 2 byte
- Default vale : 0

e) float

- Size : 4 byte
- Default value : 0.0

f) double

- Size : 8 byte
- Default value : 0.0

g) char

- Size : 16 byte
- Default value : NULL

h) boolean

- Size : 1 byte
- Default value : false

13) Arrays

An array is a group of similar type variable that are referred by the index.

a) One-dimensional array

- Syntax:-

```
data_type var_name[];
var_mn = new data _type [size];
```

- **For example,**

```
int a[];
a= new int [10];
```

- We can declare above array using following statement

```
int a[] =new int [10];
```

- In above syntax type specifies the type of data being allocated, size specifies the no of elements in the array & var_name is the array variable that is link to the array.

- The new is used to allocate an array you must specify the type & no, of elements to allocate.The array index is by default start with o.

Example – 1: abc.java

```
class abc
{
	public static void main(String s[])
	{
		int a[]={1,2,3,4,5,6};
		System.out.println(a[2]); //3
		System.out.println(a[5]); //6
	}
}
```

Example – 2: abc.java

```
class abc
{
	public static void main(String s[])
	{
		int a[] = new int[3];
		a[0] = 1;
		a[1] = 2;
		a[2] = 3;
		System.out.println(a[1]); //2
	}
}
```

b) Multi-dimentional array

- In java multi-dimensional arrays are arrays of a to declare a multi-dimensional arrays variable specify each additional index using another set of [] brackets

data_type var_nm[][] =new data_type [size] [size];

Example – 1: abc.java

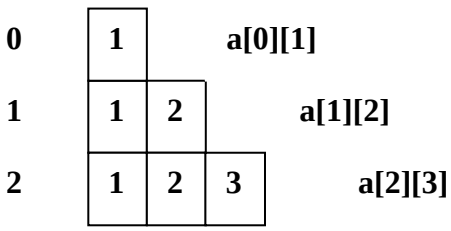
```
class abc
{
	public static void main(String s[])
	{
		int a[][] = {{1,2,3}.{2,3,4},{3,4,5}};
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				System.out.println(a[i][j]);
			}
			System.out.println("");
		}
	}
}
```

Example – 2: abc.java

```
class abc
{
	public static void main(String s[])
	{
		int k=0;
```

```
            int a[][] = new int[3][];
            a[0] = new int[1];
            a[1] = new int[2];
            a[2] = new int[3];
            for(int i=0; i<3; i++)
            {
                  for(int j=0; j<i+1; j++)
                  {
                        a[i][j] = k;
                        k++;
                  }
            }
            for(i=0; i<3; i++)
            {
                  for(j=0; j<i+1; j++)
                  {
                        System.out.print(a[i][j] + " ");
                  }
                  System.out.println("");
            }
      }
}
```

int a[] [] = new int[4] [];
Means creates 2 dimensional array and its row size is fix (4).
a[0] = new int[1];
Means we can store only one element at zero row index.
a[1] = new int[2];
Means we can store 2 elements at first row index.
a[2] = new int[3];
Means we can store 3 elements at second row index.
In general,



- You can declare array in this way…….
  Data_type [] var_name = new data_type[size];
  This declaration is know as **Alternative array Declaration.**
- int [] a = new int[3]; is equals to **int a[] = new int[3];**

14) Find length of array

In java you can find length of the array using its **length** property.

Syntax :- **array_var_name.length**

For example

```
int a[ ] = {10,20,30,40,50};
System.out.println(a.length);          //5
```

15) Operators

An operator means a symbol to perform such process like mathematical or logical operation.

For example, **a+b;** Here a and b is the operand (which is participate in process).
+ is the operator (which is use to complete this process). **a+b** is the operation (means our process).

a) Arithmetic Operator

+	:	Addition
-	:	Subtraction or unary minus
*****	:	Multiplication
/	:	Division
%	:	Modulus
++	:	Increment
--	:	Decrement

b) Assignment Operator

+=	:	Addition Assignment
-=	:	Subtraction Assignment
***=**	:	Multiplication Assignment
/=	:	Division Assignment
%=	:	Modulus Assignment

c) Relational Operator

<	:	Less then
>	:	Greater then
<=	:	Less then or equals to
>=	:	Greater then or equals to

== : Equals to
!= : Not equals to

d) Logical Operator

&& : Short circuit AND
|| : Short circuit OR
! : Logical NOT
& : Logical AND
| : Logical OR

e) Bitwise Operator

~ : Bitwise unary NOT
& : Bitwise AND
| : Bitwise OR
^ : Bitwise XOR
>> : Shift right
>>> : Shift right zero fill
<< : Shift left

f) Assignment Operator (=)

For example,

```
int x,y,z;
x = y = z = 100;
```

Means 100 is assign to x,y and z.

g) Ternary Operator (?:)

expression – 1 ? expression – 2 : expression – 3;

Above syntax the expression – 1 can be any expression that returns Boolean value. If the expression – 1 is true then the expression – 2 is executed otherwise expression – 3 is executed.

For example,

```
String s = 10 < 5 ? " Hello " : " World ";
System.out.println(s);      // World
```

16) Type Conversion & Casting

In java operation if 2 data types are compatible than java will perform the operation automatically. For example it is possible that integer value may convert into long value.

However all types are not compatible & thus all types conversion are not implicitly allow. It is also possible to obtain a conversion define from double to byte,. To do so u must use casting.

a) Automatic type conversion

When one type of data is assign to another type of variable & automatic type conversion will take place if the following 2 condition are satisfy.

(1) The 2 types are compatible
(2) The destination type is larger than the source type

For example,

```
int a;
byte b=10;
a=b;
System.out.println(a);      //10
```

b) Casting incompatible types

To create a conversion between 2 incompatible types you must have to use a cast of explicit type conversion as follows

destination _var = (target type) val/var;

For example,

```
int a = 10;
byte b = a;    //       Not possible
byte b = (byte) a;
System.out.println(b);      //10
```

17) Garbage Collection

Since objects are dynamically allocated by using the new operator you might be required to destroy such object & there memory release for later collection.

In some language like c++ dynamically allocated object must be manually released by use of a delete operated.Java takes a different approach. It handles dealocation for you automatically.

The technique that accomplices is called garbage collection where no reference to an object exist that object is assume to be no longer needed & the memory occupied by the object can be reclaimed. Garbage collection only occurs instantly during the execution of your program.

18) Finalize() method

Some times an object will need to perform some action when it is destroy.

For example if an object is holding some non java resources such as file handle or character font that you might want to make sure were this resources are remove before an object is destroyed. To handle such situation java provides a mechanism called finalization.

By using finalization you can define specific action that will occur when an object is just about to be reclaimed by the garbage collector.To add a finalzier in a class. You simply define the finalize()

```
protected void finalize ()
{
……..;
}
```

Java runtime calls finalize () whenever it is about recycle to an object of the class

19) Conditional Statements

a) if

```
if ( condition )
{
        Statements block – 1;
}
else
{
        Statements block – 2;
}
```

In above syntax condition is an expression that returns Boolean value.

If the condition is true then the Statements block – 1 is executed otherwise Statement block – 2 is executed. Here the else part is optional.

For example,

```
int i;
if(10 < 6)
{
        i=10;
}
```

```
else
{
        i=6;
}
System.out.println(i);      //6
```

b) Nested if

```
if(condition – 1)
{
        if(condition – 2)
        {
                Block – 1;
        }
        else
        {
                Block – 2;
        }
}
else
{
        Block – 3;
}
```

In above syntax condition is an expression that returns Boolean value.

Here if condition – 1 is true then JVM checks the condition – 2 and if condition – 2 is true the Block – 1 will execute otherwise Block – 2 will execute. And if condition – 1 is false the Block – 3 will execute.

For example,

```
int i;
if(10 > 6)
{
        if(5>=2)
        {
                i=1;
        }
        else
        {
                i=2;
        }
}
else
{
```

```
            i=3;
}
System.out.println(i);      //1
```

c) Else – if ladder

```
if(condition)
Block;
else if(condition)
        Block;
        else if(condition)
                Block;
        :
        :
        else if(condition)
                Block;
else
        Block;
```

As soon as one of the conditions is true, the Block associated with that if is executed, and the rest of the ladder is bypassed.

If none of the conditions is true, then the final else statement will be executed. The final else acts as a default condition; that is, if all other conditional tests fail, then the last else Block is performed. If there is no final else and all other conditions are false, then no action will take place.

For example,

```
int i;
if(10 < 5)
        i=1;
        else if(10 > 5)
                i=2;
else
        i=3;
System.out.println(i);      //2
```

d) Switch

As such, it provides a better alternative than a large series of if-else-if statements.

```
switch(expression)
{
        case literal:
                Block;
                break;
        case literal:
                Block;
                break;
                :
                :
        case literal:
                Block;
        default:
                Block;
}
```

The expression must be of type byte, short, int, or char; each of the values specified in the case statements must be of a type compatible with the expression.

Each case value must be a unique literal that is; it must be a constant, not a variable. Duplicate case values are not allowed.

The value of the expression is compared with each of the literal values in the case statements. If a match is found, the code Block following that case statement is executed. If none of the constants matches the value of the expression, then the default statement is executed.

However, the default statement is optional. If no case matches and no default is present, then no further action is taken.

For example,

```
for(char c='a';c<'d';c++)
{
        switch(c)
        {
                case 'a':
                        s="first";
                        break;
                case 'b':
                        s = "Second";
                        break;
        }
        System.out.println(s);
}
```

20) Looping Statement

The looping statements is use to repeatedly executes the same code block until the given condition will not satisfied.

a) While loop

```
while(condition)
{
        Body of loop;
}
```

The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true.

When condition becomes false, control passes to the next line of code immediately following the loop. The curly braces are unnecessary if only a single statement is being repeated.

For example

```
int i=0;
while(i < 10)
{
        System.out.println(i);
        i++;
}
```

b) do……while loop

```
do
{
        Body of loop;
}while(condition);
```

Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates. As with all of Java's loops, condition must be a Boolean expression.

The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

For example,

```
int i=0;
do
{
```

```
        System.out.println(i);
}while(i < 10);
```

c) for loop

```
for(initialization ; condition ; iteration)
{
        Body of loop;
}
```

When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop.

It is important to understand that the initialization expression is only executed once. Next, condition is evaluated. This must be a Boolean expression.

If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.

Next, the iteration portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable.

In short iteration -----> condition -----> body of loop -----> iteration and again start with condition. This process repeats until the controlling expression is false.

For example,

```
for(int i=0;i < 10;i++)
{
        System.out.println(i);
}
```

21) Branching/Jumping Statements

a) Break

In Java the break statement has three uses.

1) Terminates the code sequence in switch statement.
2) Exit from loop.
3) Use same as goto statement of C/C++.

i) Use break for exiting from loop

In loop break is use to force to terminate loop.

When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

For example,

```
for(int i=0;i < 10;i++)
{
        if(i==4) Break;
        System.out.println(i);
}
```

ii) Use break as a goto

Example: abc.java

```
class abc
{
        public static void main(String s[])
        {
                lbl1:
                {
                        lbl2:
                        {
                                System.out.println(" Hello ");
                                if(10 > 5) break lbl1;
                                System.out.println(" Not executed ");
                        }
                        System.out.println(" lbl1 is here ");
                }
        }
}
```

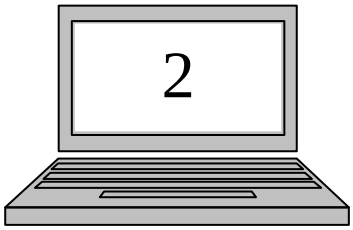
Keep in mind that you can not use this version of break in different scope. i.e. the label and the break must be in same scope.

b) continue

In while and do-while loops, a continue statement causes control to be transferred directly to the conditional expression that controls the loop. In a for loop, control goes first to the iteration portion of the for statement and then to the conditional expression.

For example,

```
for(int i=0;i<10;i++)
{
        if(i%2==0) continue;
        System.out.println(i);
}
```



CHAPTER TWO

Class Fundamental

IN THIS CHAPTER

- Creating Class
- Declaring an Object
- Methods
- Constructor
- Method Overloading
- Passong Object as Argument
- Returning Object From Function
- Static Members
- This Keyword
- Command Line Argument
- Inheritance
- Super Keyword
- Method Overriding
- Final Keyword
- Nested Class
- Dynamic Method Dispatch
- Abstract Keyword
- Interface
- Access Specifiers

1) Creating class

The class is the core of the object oriented language in java. Any concept you wish to implement in a java program must be encapsulated within a class.

The most important things to be understand about a class is that it defines a new data type using the new type can be create object of that type. Thus, class is like a template for creating of its object.

The general form of class are as follow.

```
class class_name
{
        type variable_1;
        type variable_2;
        :
        :
        type variable_N;
        type methodName1(parameter_list)
        {
                //Body of the method 1
        }
        type methodName2(parameter_list)
        {
                //Body of the method 2
        }
        type methodNameN(parameter_list)
        {
                //Body of the methodN
        }
}
```

The data or variables defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class.

2) Declaring an object

When you create a class, you are creating a new datatype. You can use this type to create a variable of that class type.

If you want to create an object then you have to give physical memory that equivalent to class (object) and assign it to that variable. You can do this using the **new** operator. The new operator allocates memory for an object.

The general form of declaring an object is:

class_name object_nm = **new** class_name();

Example: abc.java

```
class A
{
        int i,j,k;
}
class abc
{
        public static void main(String s[])
        {
                A a1 = new A();                 // creating an object named a1.
                a1.i = 10;
                a1.j = 20;
                a1.k = 30;
                System.out.println(a1.i+","+a1.j+","+a1.k);   //10,20,30
                A a2 = new A();                 // creating an object named a2.
                a2.i = 100;
                a2.j = 200;
                a2.k = 300;
                System.out.println(a2.i+","+a2.j+","+a2.k);   //100,200,300
        }
}
```

On the basis of above program we can say that the every object of the class has separate copy of the member variables of that class.

3) Methods

The method is just a name of code block, which we can further use that code block to just calling that name. Thus it decreases the code lines of our program.

A general form of method is:

return_type method_name(parameter_list)
{
Code block;
}

If the method does not return any value then set it to **void.** The name of the method is specified by method_name. This can be any legal identifier other than those already used by other items within the current scope.

The parameter-list is a sequence of type and identifier pairs separated by commas. Parameters are variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

Methods that have a return type other than void, return a value to the calling routine using the return statement

return value/variable;

Here, value/variable is the value to be returned.

Example – 1: abc.java

```
class A
{
	int i,j;
	void get_d(int i1,int j1)
	{
		i=i1;
		j=j1;
	}
	void disp()
	{
		System.out.println(i+”,”+j);
	}
}
class abc
{
	public static void main(String s[])
	{
		A a1 = new A();	// creating an object named a1.
		a1.get_d(10,20);
		a1.disp();		//10,20
	}
}
```

Example – 2: abc.java

```
class A
{
        int i,j;
        void get_d(int i1,int j1)
        {
                i=i1;
                j=j1;
        }
        int sum()
        {
                int s = i+j;
                return s;
        }
}
class abc
{
        public static void main(String s[])
        {
                A a1 = new A();                 // creating an object named a1.
                a1.get_d(10,20);
                int a=a1.sum();
                System.out.println(" Sum : " + a);                  // 30
        }
}
```

4) Constructor

A constructor is use to initialize the object at the creation time. And it is a special type of member function.It is compulsory to set name of the constructor as same as the class name in which the constructor is defined.

A constructor is implicitly (automatically) called when the object is created. It has no return type, not even void. A constructor can have arguments. Thus we can overload the constructor.A general form of constructor is:

```
        class class_name
        {
                class_name(){……..}
                class_name(parameter_list)
                {…….}
        }
```

Example: abc.java

```
class A
{
	int i,j;
	A()
	{
		i=10;
		j=20;
	}
	A(int i1,int j1)
	{
		i=i1;
		j=j1;
	}
	A(A a)
	{
		i=a.i;
		j=a.j;
	}
	void disp()
	{
		System.out.println(i+”,”+j);
	}
}
class abc
{
	public static void main(String s[])
	{
		A a1 = new A();
		A a2 = new A(100,200);
		A a3 = new A(a1);
		a1.disp();		//10,20
		a2.disp();		//100,200
		a3.disp();		//10,20
	}
}
```

5) Method overloading

In java we can define more then one method in same scope which has same name. This method is known as overloaded methods and this process is known as overloading.

The method overloading concept is known as polymorphism which means **“one name many form”**.

The method overloading is depending on either total number of arguments or different types of arguments. Thus java executes that overloaded method whose arguments match whose parameters match the arguments used in the call.

Example: abc.java

```
class A
{
	void disp()
	{
		System.out.println(“ No argument “);
	}
	void disp(int i)
	{
		System.out.println(“ int type single argument “);
	}
	void disp(String s,int i)
	{
		System.out.println(“ more then one argument – String & int “);
	}
	void disp(String s)
	{
		System.out.println(“ String type single argument “);
	}
	void disp(int i,String s)
	{
		System.out.println(“ more then one argument – int & String “);
	}
}
class abc
{
	public static void main(String s[])
	{
```

```
        A a1 = new A();
        a1.disp();              // No argument
        a1.disp("Hello");       // String type single argument
        a1.disp(10);            // int type single argument
        a1.disp("Hello",10);    // more then one argument – String & int
        a1.disp(10,"Hello");    // more then one argument – int & String
    }
}
```

6) Passing object as argument

We can pass object as an argument in method and constructor.

When we pass object as an argument the all instance variable copy of that object is also pass to method.

Example: abc.java

```
class A
{
    int i;
    void get_d(int i1)
    {
        i=i1;
    }
    void check(A a)
    {
        if(i < a.i)
        {
            System.out.println( i +" is minimum ");
        }
        else
        {
            System.out.println( a.i + " is minimum ");
        }
    }
}
class abc
{
    public static void main(String s[])
    {
        A a1 = new A();
```

```
            A a2 = new A();
            a1.get_d(10);
            a2.get_d(20);
            a1.check(a2);
        }
}
```

7) Returning object from function

Example : abc.java

```
class A
{
        int i,j;
        void get_d(int i1,int j1)
        {
                i=i1;
                j=j1;
        }
        void disp()
        {
                System.out.println(i+",”+j);
        }
        A change(A a)
        {
                a.i = a.i+i;
                a.j = a.j+j;
        }
}
class abc
{
        public static void main(String s[])
        {
                A a1 = new A();
                A a2 = new A();
                a1.get_d(100,200);
                a1.disp();                      //100,200
                a2=a1.change(a1);
                a2.disp();                      //200,400
        }
}
```

8) Static members

In java normally when we declare any non static member variable at that time java has not allocate memory to store value.

Java allocates memory to that variable when we create an object of the class in which that non static member variable is declared.

But if we want to allocate the memory at declaration time then the static must be use.

In java there are 2 use of static key word.

1) static member variable 2) static member function.

a) Static member variable

We can declare static member variable using static key word are as follows:

static data_type variable=value;

It is important to note that when the static member variable is declared at that time it must be initialized.

The difference between static and non static variable is that every object has their own copy of non static member variable while only one copy of static member variable is shared between every object.

To initialize the static member variable there are 2 way in java.

1) Initialized it at declaration time. 2) Use static block.

Example – 1:abc.java

```
class A
{
	static int i=10;
	static int j=20;
}
class abc
{
	public static void main(String s[])
	{
		System.out.println(A.i+A.j);
	}
}
```

Example – 2:abc.java

```
class A
{
        static int i,j;
}
class abc
{
        public static void main(String s[])
        {
                A.i=10;
                A.j=20;
                System.out.println(A.i+A.j);
        }
}
```

b) Using Static block

Example – 3:abc.java

```
class A
{
        static int i,j;
        static
        {
                i=10;
                j=20;
        }
}
class abc
{
        public static void main(String s[])
        {
                System.out.println(A.i+A.j);
        }
}
```

Example – 4:abc.java

```
class A
{
        static int i,j;
}
class abc
{
```

```
	static
	{
		A.i=10;
		A.j=20;
	}
	public static void main(String s[])
	{
		System.out.println(A.i+A.j);
	}
}
```

c) Static member function

The static member function can access only static variable.
It calls with the class name in which it is defined but not with the object.
It calls only other static function in it.

Example – 1:abc.java

```
class abc
{
	static void disp()
	{
		System.out.println(" Hello ");
	}
	public static void main(String s[])
	{
		disp();
	}
}
```

Example – 2:abc.java

```
class A
{
	static void disp()
	{
		System.out.println(" Hello ");
	}
}
class abc
{
	public static void main(String s[])
	{
```

```
                A.disp();
        }
}
```

Example – 3:abc.java

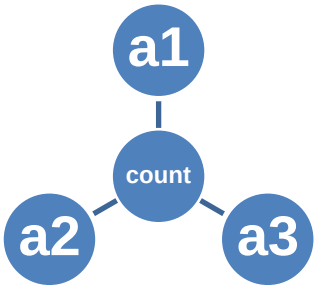
```
class A
{
        static int i=10,j=20;
        static void disp()
        {
                System.out.println(i+j);
        }
}
class abc
{
        public static void main(String s[])
        {
                A.disp();
        }
}
```

Example – 4:abc.java

```
class A
{
        int i;
        static int count=0;
        void get_d(int i1)
        {
                i=i1;
                count++;
        }
        void disp_d()
        {
                System.out.println(i);
        }
        static void disp_count()
        {
                System.out.println(count);
        }
}
class abc
{
```

```
        public static void main(String s[])
        {
                A a1 = new A();
                A a2 = new A();
                A a3 = new A();
                a1.get_d(10);
                System.out.println(a1.count);   //1
                a2.get_d(20);
                System.out.println(a2.count);   //2
                a3.get_d(30);
                System.out.println(a3.count);   //3
                a1.disp_d();                    //10
                a2.disp_d();                    //20
                a3.disp_d();                    //30
                System.out.println(a1.count);   //3
                System.out.println(a2.count);   //3
                System.out.println(a3.count);   //3
                A.disp_count();                 //3
        }
}
```

- The following figure shows you that how the static variable works in above program.



9) This Keyword

There are 2 use of this keyword:

1) To resolve name-space collisions in methods.
2) To refer the current object.

Example – 1:abc.java

```
class A
{
        int i,j;
        void get_d(int i,int j)
        {
```

```
                this.i = i;
                this.j = j;
        }
        void disp()
        {
                System.out.println(i+”,”+j);
        }
}
class abc
{
        public static void main(String s[])
        {
                A a1 = new A();
                a1.get_d(10,20);
                a1.disp();
        }
}
```

Example – 2:abc.java

```
class A
{
        int i,j;
        void get_d(int i1,int j1)
        {
                i=i1;
                j=j1;
        }
        void disp()
        {
                System.out.println(i+","+j);
        }
        A change(int chg)
        {
                i=i+chg;
                j=j+chg;
                return this;
        }
}
class abc
{
        public static void main(String s[])
        {
```

```
            A a1 = new A();
            A a2 = new A();
            a1.get_d(10,20);
            a1.disp();
            a2=a1.change(15);
            a2.disp();
        }
}
```

10) Command line argument

If you want to pass information into a program when you run it is accomplish by passing command line arguments to main()

The command line argument is the info that directly follows the programs name on the command line argument inside a java program is possible by s[ ] of type String.

The first command line argument is stores in s[0] second at s[1] & so ..on.

Each command line argument is separated by space.

Example:abc.java

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

11) Inheritance

Re-usability is the main aspect of OOP paradigm. It is always good if we could reuse something that already exist rather than creating the same all over again

Java class can be re-used in several ways. This is basically done by creating new classes reusing the properties of existing one.

The mechanism of driving a new class from an older one is called inheritance and the older one class is known as base/super/parent class & the new class is known as derived/sub/child class The inheritance allows sub classes to inherit all the variable & methods of the super class.

In java we can not extends more then one class at a time. It means that in java that inheritance is possible which has one and only one super class.

In java we can create inheritance between 2 classes use **extends** keyword.

```
class sub_class_name extends super_class_name
{
        ……….;
        ……….;
}
```

Example – 1:abc.java

```
class A
{
        void disp()
        {
                System.out.println(" A class ");
        }
}
class B extends A
{
        void show()
        {
                System.out.println(" B class ");
        }
}
class abc
{
        public static void main(String s[])
        {
                B b1 = new B();
                b1.disp();                      //A class
                b1.show();                      //B class
        }
}
```

Example – 2:abc.java

```
class A
{
        int i,j;
        void get_d(int i1,int j1)
        {
                i=i1;
                j=j1
        }
}
class B extends A
{
        void disp()
        {
                System.out.println(i+“,”+j);
        }
}
class abc
{
        public static void main(String s[])
        {
                B b1 = new B();
                b1.get_d(10,20);
                b1.disp();                              //10,20
        }
}
```

Example – 3:abc.java

```
class A
{
        int i,j;
        void get_d(int i1,int j1)
        {
                i=i1;
                j=j1
        }
}
class B extends A
{
        void disp()
        {
                System.out.println(i+“,”+j);
```

```
        }
}
class C extends B
{
        void change()
        {
                i=i+10;
                j=j+10;
        }
}
class abc
{
        public static void main(String s[])
        {
                C c1 = new C();
                c1.get_d(10,20);
                c1.disp();                      //10,20
                c1.change();
                c1.disp();                      //20,30
        }
}
```

12) super keyword

In java the super keyword has 2 uses:

1) To call super class constructor.
2) To access member of super class that has been hidden by member of sub class.

a) Call super class constructor

We can call super class related constructor are as following form of super:

super(parameter_list);

The super() is always placed at the sub class constructor's first line.

Example: abc.java

```
class A
{
        int i,j;
        A(int i1,int j1)
```

```
	{
		i=i1;
		j=j1;
	}
}
class B extends A
{
	int k;
	B(int x,int y,int z)
	{
		super(x,y);
		k=z;
	}
	void disp()
	{
		System.out.println(i+","+j+","+k);
	}
}
class abc
{
	public static void main(String s[])
	{
		B b1 = new B(10,20,30);
		b1.disp();
	}
}
```

b) Access member of super class that has been hidden by member of sub class.

We can access the super class member (member variable or member function) that has been hiding by member of subclass using following form of super keyword.

super.members;

Example: abc.java

```
class A
{
	int I;
}
class B extends A
{
	int i;
	void get_d()
```

```
	{
		super.i=10;
		i=20;
	}
	void disp()
	{
		System.out.println(" Super class I : "+super.i+", and Sub class I : "+i);
	}
}
class abc
{
	public static void main(String s[])
	{
		B b1 = new B();
		b1.get_d();
		b1.disp();
	}
}
```

13) Method Overriding

Method overriding is performing in different class in inheritance.

The method overriding means to define method in sub class which has name, return type and argument list as same as super class any method.

In short method overriding is the process to give different definition of the same method in different class.

Example:abc.java

```
class A
{
	int i;
	void get_d(int i1)
	{
		i=i1;
	}
}
class B extends A
{
	int j;
```

```
        void get_d(int j1)
        {
                super.get_d(j1);
                j=j1+10;
        }
        void disp()
        {
                System.out.println(i+",,+j);
        }
}
class abc
{
        public static void main(String s[])
        {
                B b1 = new B();
                b1.get_d(10);
                b1.disp();                              //10,20
        }
}
```

14) final keyword

In java the final keyword has 3 uses:

1) To create constant variable
2) To prevent inheritance.
3) To prevent method overriding

a) Create constant using final

In C/C++ we can create constant using **const** keyword, but in java to create constant we use final keyword are as follows:

final data_type var_name=value;

For example,

```
final int i=10;
```

b) Prevent inheritance

If we want to restrict to create inheritance just declare super class as final.

```
final class super_class_name
{
        ……;
}
class sub_class_name extends super_class_name
```

The above inheritance is not possible because of the super class declared as final.

c) Prevent method overriding

If we want to restrict method overriding/method redefine then just declare that method as final which is override in sub class.

15) Nested class

The nested class means to define class in scope of other class. It means nested class is a defining class in another class.

Example:abc.java

```
class A
{
	int i=10,j=20;
	class B
	{
		int k=30;
	}
	void disp()
	{
		B b = new B();
		System.out.println(i+","+j+","+b.k);
	}
}
class abc
{
	public static void main(String s[])
	{
		A a1 = new A();
		a1.disp();			//10,20,30
	}
}
```

16) Dynamic method dispatch

It is a mechanism by which a call to overridden method at run time rather than compile time.

Super class reference variable can refer to a sub-class object. Java uses this technology to calls overridden methods at runtime

When an overridden method is call to the super class reference, Java determines which version of that method to execute base on the type of object being referred at run time call occurs. This is made at run time.

Example:abc.java

```
class A
{
        void disp()
        {
                System.out.println(" Hello ");
        }
}
class B extends A
{
        void disp()
        {
                System.out.println(" Hi ");
        }
}
class C extends B
{
        void disp()
        {
                System.out.println(" Bye ");
        }
}
class abc
{
        public static void main(String s[])
        {
                A a1;
                A a = new A();
                B b = new B();
                C c = new C();
                a1=a;
                a1.disp();                      //Hello
                a1=b;
                a1.disp();                      //Hi
                a1=c;
                a1.disp();                      //Bye
        }
}
```

17) abstract keyword

In java we can create abstract class and abstract method using abstract keyword.

Sometimes we will want to define a class that declares a method's structure to without giving its definition. And its sub classes give that declared method definition in it. This is done by abstract method.

An abstract method must be declared in abstract class. But abstract class can contain normal methods too.

And it is responsibility of sub class that they must be defining all abstract method of their super class in them.

You are not to create an object of abstract class. You can create a reference of abstract class. You can not create abstract constant, abstract static method.

You can create an abstract class and abstract methods are as follows:

```
abstract class class_name
{
        abstract return_type method_name(argument_list);
        abstract return_type method_name(argument_list);
                                :
                                :
        abstract return_type method_name(argument_list);
        return_type method_name(argument_list)
        {
                ……..;
                ……..;
        }
}
```

Example - 1:abc.java

```
abstract class A
{
        void disp1()
        {
                System.out.println(" Hello from disp – 1A ");
        }
        abstract void disp2();
        abstract void disp3();
```

```
}
class B extends A
{
	void disp2()
	{
		System.out.println(" Hello from disp – 2B ");
	}
	void disp3()
	{
		System.out.println(" Hello from disp – 3B ");
	}
}
class abc
{
	public static void main(String s[])
	{
		B b1 = new B();
		b1.disp1();
		b1.disp2();
		b1.disp3();
	}
}
```

Example – 2:abc.java

```
abstract class A
{
	void disp1()
	{
		System.out.println(" Hello from disp – 1A ");
	}
	abstract void disp2();
	abstract void disp3();
}
class B extends A
{
	void disp2()
	{
		System.out.println(" Hello from disp – 2B ");
	}

	void disp3()
	{
```

```
                System.out.println(" Hello from disp – 3B ");
        }
}
class C extends A
{
        void disp2()
        {
                System.out.println(" Hello from disp – 2C ");
        }

        void disp3()
        {
                System.out.println(" Hello from disp – 3C ");
        }
}
class abc
{
        public static void main(String s[])
        {
                B b1 = new B();
                C c1 = new C();
                b1.disp1();             // Hello from disp – 1A
                c1.disp1();             // Hello from disp – 1A
                b1.disp2();             // Hello from disp – 2B
                c1.disp2();             // Hello from disp – 2C
                b1.disp3();             // Hello from disp – 3B
                c1.disp3();             // Hello from disp – 3C
        }
}
```

18) Interface

In java we can not inherit more then one class but we can inherit more then one interface.

Only methods declaration is allowed in interface. Thus we can say that the interface is the fully abstract class because an abstract class allows defining normal methods in it while interface not allowed.

Using interface we can specify that what a class must do but not how it does.

Interfaces are similar to classes but they lack instance variable & there methods are declare without any body. It means you can define interface but you do not assumption how they are implemented.

Once it is define, any number of classes can use an interface & also one class can use any number of interfaces like multiple inheritance.

By default every method of interface is abstract. Thus there is no need to writer abstract keyword.

We can define an interface using **interface** keyword are as follows:

```
access_specifier interface interface_name
{
        static/final data_type var_name = value;
        return_type method_name(argument_list);
}
```

Variable declare inside the interface are implicitly final or static that is they can not be changed. And they must be initialized. Methods of interface are implicitly public.

In previous we create an inheriting class using **extends** keyword, in same way we can inherit interface using **implements** keyword are as follows:

```
class  class_name implements interface_name1,interface_name2,…
```

If in our program one class needs to inherit both class and interface then follows the following:

```
class class_name extends super_class_name implements intercace_name
```

If in our program one interface needs to inherit another interface then follows the following: `interface interface_name1 extends intercace_name2`

In short just remember that

- class **extends** class.
- class **implements** interface.
- interface **extends** interface

In java one interface can not extends/implements a class.

The class which implements the interface has the responsibility to define methods of the implementing interface.

Example – 1:abc.java

```
interface A
{
        void disp1();
        void disp2();
}
class B implements A
{
        public void disp1()
        {
                System.out.println(" Hello ");
        }
        public void disp2()
        {
                System.out.println(" World ");
        }
}
class abc
{
        public static void main(String s[])
        {
                B b1 = new B();
                b1.disp1();
                b1.disp2();
        }
}
```

Example – 2:abc.java

```
interface A
{
        void disp1();
}
interface B
{
        void disp2();
}
class C implements A,B
{
        public void disp1()
        {
                System.out.println(" Hello ");
        }
```

```
	public void disp2()
	{
		System.out.println(" World ");
	}
}
class abc
{
	public static void main(String s[])
	{
		C c1 = new C();
		c1.disp1();
		c1.disp2();
	}
}
```

Example – 3:abc.java

```
interface A
{
	void disp1();
}
class B
{
	void disp2()
	{
		System.out.println(“ World “);
	}
}
class C extends B implements A
{
	public void disp1()
	{
		System.out.println(“ Hello “);
	}
}
class abc
{
	public static void main(String s[])
	{
		C c1 = new C();
		c1.disp1();
		c1.disp2();
	}
}
```

Example – 4:abc.java

```
interface A
{
        void disp1();
}
interface B extends A
{
        void disp2();
}
class C implements B
{
        public void disp1()
        {
                System.out.println(" Hello ");
        }
        public void disp2()
        {
                System.out.println(" World ");
        }
}
class abc
{
        public static void main(String s[])
        {
                C c1 = new C();
                c1.disp1();
                c1.disp2();
        }
}
```

We can create a class, in which the interface is defined its call nested interface.

Example – 5:abc.java

```
class A
{
        interface B
        {
                void disp1();
        }
        void disp2()
        {
                System.out.println(" World ");
```

```
	}
}
class C extends A implements A.B	// or class C extends A
{
	public void disp1()
	{
		System.out.println(" Hello ");
	}
}
class abc
{
	public static void main(String s[])
	{
		C c1 = new C();
		c1.disp1();
		c1.disp2();
	}
}
```

If a class implements an interface but does not full implements into the method define by that interface than that class must be declare as abstract. This is known as partial implementation of interface.

Example – 6:abc.java

```
interface A
{
	void disp1();
	void disp2();
}
abstract class B implements A
{
	public void disp1()
	{
		System.out.println(“ Hello “);
	}
}
class C extends B
{
	void disp3()
	{
		System.out.println(“ World “);
	}
```

```
}
class abc
{
	public static void main(String s[])
	{
		C c1 = new C();
		c1.disp1();
		c1.disp3();
	}
}
```

19) Access Specifiers

In java there are 4 types of access specifiers.

d) private

When we declare any methods, variable or class as private it is accessed in the same scope of the file.

e) protected

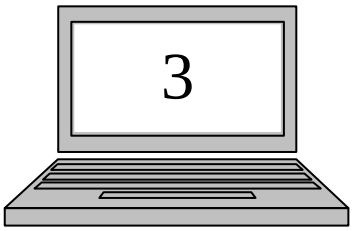
When we declare any methods, variable or class as protected it is accessed in the same class and its sub class of the file.

f) public

When we declare any methods, variable or class as public it is accessed any where in the system (i.e. same package or another package in our system).

g) no modifier

This access specifier is work as same as public with the small difference that, When we declare any methods, variable or class as public it is accessed only in same package.



CHAPTER THREE

Packages

IN THIS CHAPTER

- Packages

1)Packages

Simple meaning of the package is a directory or folder of our system.

If in our program we need to create more then one java file with same name but our operating system does not allow us to create more then one file which has same name and extension in same directory. This problem solved by creating Package.

The package has both a naming & visibility control mechanism; you can define classes and interface inside a package.

a) Creating a package

To create a package simply include a package statement as first line in a java source file

Any classes declared within that file will belong to the specified package. The package statement defines a name space in which classes is stored.

If you omit the package statement the class names are put into default package which has no name.

The general form for package statement is as follows

package pkg_name;

Remember that the package name and the directory name in which we are saved our program are same.

You can create a hierarchy of packages to do so simply separate each package name from one by using of a period (.)

package [p1] [.p2] [.p3];

b) Compiling and executing a package

Consider the following program to understand the compilation and execution of package.

Assume this path ---- **c:\programs**

```
package p1;
class abc
{
```

```
        public static void main(String s[])
        {
            System.out.println("Hello");
        }
    }
```

Now, create folder in programs directory which has same name as package name.

i.e. **c:\programs\p1**

And save this program in p1 folder which has same name of the class.

i.e. **c:\programs\p1\abc.java**

If in our file there are more than one class is defined than save that file named of any class name. But if there is a main class (i.e. a class which has main() method) in the file then save that file with name of the class which has main() method.

Now go to **c:\programs\p1** in command prompt and write following command to compile package program.

c:\programs\p1\>javac abc.java

You can also compile as follows. (Batter option to compile package programs)

c:\programs\p1\>javac *.java

or

c:\programs\>javac p1*.java

To execute the package program first go to the parent directory of the package which we want to execute and write following.

c:\programs\>java p1.abc

Example – 1:

c:\program\p1\A.java

```
package p1;
class A
{
    void disp()
    {
        System.out.println(" Hello ");
    }
}
```

c:\program\p1\abc.java

```
package p1;
class abc
{
	public static void main(String s[])
	{
		new A().disp();
		/*
			A a1 = new A();
			a1.disp();
		*/
	}
}
```

- Compile – **c:\programs\>javac p1*.java**
- Execute - **c:\programs\>java p1.abc**

Example – 2:

c:\programs\p1\A.java

```
package p1;
class A
{
	int I,j;
	void get_d(int i1,int j1)
	{
		i=i1;
		j=j1;
	}
}
```

c:\programs\p1\B.java

```
package p1;
class B extends A
{
	void disp()
	{
		System.out.println(i+”,”+j);
	}
}
```

c:\programs\p1\abc.java

```
package p1;
class abc
{
	public static void main(String s[])
```

```
	{
		B b1 = new B();
		b1.get_d(10,20);
		b1.disp();
	}
}
```

- Compile – **c:\programs\>javac p1*.java**
- Execute – **c:\programs\>java p1.abc**

Example – 3:

c:\programs\p1\A.java

```
package p1;
interface A
{
	void disp1();
	void disp2();
}
```

c:\programs\p1\B.java

```
package p1;
class B implements A
{
	public void disp1()
	{
		System.out.println(" Hello ");
	}
	public void disp2()
	{
		System.out.println(" World ");
	}
}
```

c:\programs\p1\abc.java

```
package p1;
class abc
{
	public static void main(String s[])
	{
		new B().disp1();
		new B().disp2();
	}
}
```

- Compile – **c:\programs\>javac p1*.java**
- Execute – **c:\programs\>java p1.abc**

Example – 4:

c:\programs\p1\A.java

```
package p1;
public class A
{
        public void disp()
        {
                System.out.println(" Hello ");
        }
}
```

c:\programs\p2\abc.java

```
package p2;
class abc
{
        public static void main(String s[])
        {
                new p1.A().disp();
                /*      p1.A a1 = new p1.A();
                        a1.disp();      */
        }
}
```

- Compile – **c:\programs\>javac p1*.java**
  c:\programs\>javac p2*.java
- Execute - **c:\programs\>java p2.abc**

Example – 5:

c:\programs\p1\A.java

```
package p1;
public class A
{
        public void disp1()
        {
                System.out.println(" Hello ");
        }
}
```

c:\programs\p2\B.java

```
package p2;
public class B extends p1.A
{
        public void disp2()
        {
                System.out.println(" World ");
```

```
        }
}
```

c:\programs\p3\abc.java

```
package p3;
class abc
{
        public static void main(String s[])
        {
                p2.B b1 = new p2.B();
                b1.disp1();
                b1.disp2();
        }
}
```

- Compile – **c:\programs\>javac p1*.java**
  c:\programs\>javac p2*.java
  c:\programs\>javac p3*.java
- Execute – **c:\programs\>java p3.abc**

c) Importing a package

We can call the whole java file of another package in another package using importing package.

In a java source file, import statement occur immediately after package statement & before the class definition

import pkg1[.pkg2][.pkg3][.class_nm/.*];

In above syntax you can specify a particular class name or put * to import all classes of the package.

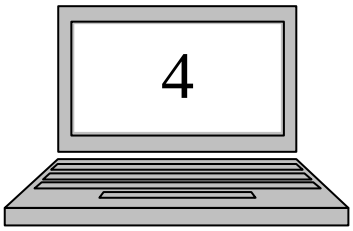
Example – 6:

c:\programs\p1\A.java

```
package p1;
public class A
{
        public void disp()
        {
                System.out.println(" Hello ");
        }
}
```

c:\programs\abc.java

```
import p1.*;
class abc
{
	public static void main(String s[])
	{
		A a1 = new A();
		a1.disp();
	}
}
```



CHAPTER FOUR

Exception Handling

IN THIS CHAPTER

- Exception Handling

1) Exception Handling

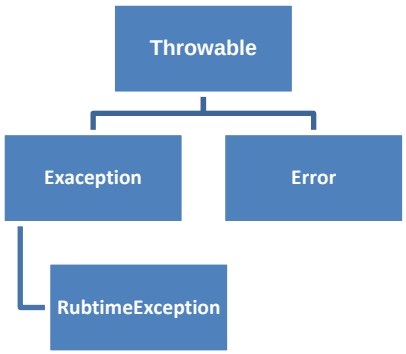
An exception means the run time error. It means when we execute the program at that time some errors may generate. This error calls exception. And to handle this exception we must use exception handling mechanism.

The main aim of exception handling mechanism is to display user friendly error message instead of java's built in message and stop the unconditionally terminating the program.

It is important to know that the error and an exception are different.

An error is generating at compile time and we could not handle it, while exception is generating at run time and we can handle it.

In java the Throwable is the super class and it has 2 sub classes that are Exception and Error.



A java exception is the object that describes an exceptional conditional that has occurred in a code sequence.

When an exception condition arise an object representing an exception is created & throw in the method at paused the error.

Exception can be generated by java runtime system or it may be generated manually by a code. Manually generated exceptions are typically used to report some error condition to the caller of a method.

Java exception handling manages by 5 keywords

(1) try
(2) catch

(3) throw
(4) throws
(5) finally

In our program some code block may be generate the exception then put that code within a **try** block

If an exception occurs within the try block and that exception is system generated exception then it automatically thrown by java runtime system. And To manually throw exception use the keyword **throw**

A **catch** block is use to catch the thrown exception from the try block to handle it.

If in a method there may be generate the exception than that method throw generated exception using **throws** keyword.

Any code that must be executed after a try block then put in a **finally** block.

Exception is the class which is used for exceptional conditions that user program should catch exception is also the class that you will sub-class to create your custom exception type

There is an important sub-class of exception called RuntimeException. Errors of this type are automatically defining program that you write such as division by zero or invalid array indexing or many more.

For example,

```
class abc
{
	public static void main(String s[])
	{
		System.out.println(“ Division is : “+(10/0));
	}
}
```

When the java runtime system detects the divide by zero it creates a new exception object & then throw this exception object.

In above program the execution of abc to stop because once an exception has been thrown it must be catch by an exception hanlder.

This exception catches by the default handler provided by java runtime system. Default handler displays a string describing the exception. This string is known as **StackTrace**

For example,

Exception in thread "main" java.lang.ArithmeticException: / by zero
at abc.main(abc.java:5)

It means StackTrace displays following…

- Fully qualified exception class name (**java.lang.ArithmeticException**)
- Exception name (**/ by zero**)
- Class and function name in which the exception is generated. (**abc.main**)
- File name and line number where generate the exception. (**abc.java:5**)

a) try and catch block

In our program some codes generate the exception than put it in try block. In try block when exception is generate it is throw to the catch to handle it.

Note that it is compulsory to put catch or finally block immediately after a try block is over.

One try can have one or more than one catch but more than one try can not have only one catch. It means every try has minimum one catch.

```
try
{
        ……;
}
catch(exception_class_name object_name)
{
        ……..;
}
```

Once an exception is thrown program control transfer from the try block to the catch block. Catch is never called so execution never returns to the try blocks form a catch.

Once a catch statement has executed program control continue with the next line in the program.

A catch statement can not catch an exception thrown by another try statement.

Example – 1:abc.java

```
class abc
{
        public static void main(String s[])
        {
                try
```

```
		{
			System.out.println(10/0);
		}
		catch(ArithmeticException e)
		{
			System.out.println("Exception : Divide by zero ");
			System.out.println(“ Hello from catch “);
		}
		System.out.println(“ Hello “);
	}
}
```

Example – 2:abc.java

```
class abc
{
	public static void main(String s[])
	{
		int m;
		for(int i=0;i<15;i++)
		{
			try
			{
				m=10/(i-10);
				System.out.println(i);
			}
			catch(ArithmeticException e)
			{
				System.out.println(" Divide by zero ");
			}
		}
	}
}
```

Example – 3:abc.java

```
class abc
{
	public static void main(String s[])
	{
		try
		{
			System.out.println(10/s.length);
		}
```

```
            catch(ArithmeticException e)
            {
                System.out.println(" Divide by zero ");
            }
        }
}
```

b) Multiple catch for one try

Example – 4:abc.java

```
class abc
{
    public static void main(String s[])
    {
        int s[] = new int[5];
        int m;
        for(int i=0;i<10;i++)
        {
            try
            {
                m=10/(i-2);
                s[i]=i*10;
                System.out.println(i);
            }
            catch(ArithmeticException e)
            {
                System.out.println(" divide by o ");
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                System.out.println(" array overflow ");
            }
        }
    }
}
```

c) Nested try

Example – 5:abc.java

```
class abc
{
    public static void main(String s[])
```

```
	{
		int s[] = new int[5];
		int m;
		for(int i=0;i<10;i++)
		{
			try
			{
				try
				{
					m=10/(i-2);
					s[i]=i*10;
					System.out.println(i);
				}
				catch(ArithmeticException e)
				{
					System.out.println(" divide by o ");
				}
			}
			catch(ArrayIndexOutOfBoundsException e)
			{
				System.out.println(" array overflow ");
			}
		}
	}
}
```

In java exception handling mechanism all types of Exception catches in following catch.

```
	catch(Exception e)
	{
		……;
	}
```

Example – 6:abc.java

```
class abc
{
	public static void main(String s[])
	{
		try
		{
			System.out.println(10/0);
		}
		catch(Exception e)
```

```
            {
                System.out.println("Exception : “+e);
            }
        }
}
```

d) throw clause

It is possible for your program to throw an exception explicitly using the throw statement

The general form of throw statement is:
throw throwable_instance;

Here throwable_instance must be an object of type Throwable or a sub-class of Throwable.

Primitive type such as int, char as well as non-throwable classes such as String, Object can not be used as exception.

There are 2 ways you can obtain a throwable object

(1) using a parameter in a catch clause
(2) creating one with the new operator

The execution stops immediately after the throw statement and any sub-sequence statement are not executed.

Example – 7:abc.java

```
class abc
{
    public static void main(String s[])
    {
        for(int i=0;i<10;i++)
        {
            try
            {
                if(i-3==0)
                {
                    throw new ArithmeticException();
                }
                else
                {
                    System.out.println(i);
```

```
                }
            }
            catch(ArithmeticException e)
            {
                System.out.println(" divide by 0 ");
            }
        }
    }
}
```

Example – 8:abc.java

```
class abc
{
    public static void main(String s[])
    {
        for(int i=0;i<10;i++)
        {
            try
            {
                if(i-3==0)
                {
            throw new ArithmeticException(“ Divide by zero : Hello “);
                }
                else
                {
                    System.out.println(i);
                }
            }
            catch(ArithmeticException e)
            {
                System.out.println(e);
            }
        }
    }
}
```

e) throws clause

A throws clause lists the types of exceptions that a method might throw. This is necessary for all exceptions, except those of type Error or RuntimeException, or any of their subclasses.

All other exceptions that a method can throw must be declared in the throws clause. If they are not, a compile-time error will result.

General from of method is:

```
return_type method_name(parameter-list) throws exception1,….,exceptionN
{
    // body of method
}
```

Exception – 9:abc.java

```
class abc
{
    static void disp()
    {
        System.out.println(10/0);
    }
    public static void main(String s[])
    {
        try
        {
            disp();
        }
        catch(ArithmeticException e)
        {
            System.out.println(" Divide by zero ");
        }
    }
}
```

Example – 10:abc.java

```
class abc
{
    static void disp()
    {
        int m;
        for(int i=0;i<10;i++)
        {
            m=10/(i-5);
            System.out.println(i);
        }
    }
    public static void main(String s[])
```

```
	{
		try
		{
			disp();
		}
		catch(ArithmeticException e)
		{
			System.out.println();
		}
	}
}
```

Example 11:abc.java

```
class A
{
	void disp() throws IllegalAccessException
	{
		for(int i=0;i<10;i++)
		{
			if(i-3==0)
			{
				throw new IllegalAccessException();
			}
			else
			{
				System.out.println(i);
			}
		}
	}
}
class abc
{
	public static void main(String s[])
	{
		try
		{
			new A().disp();
		}
		catch(Exception e)
		{
			System.out.println(e);
		}
```

```
        }
}
```

There are 2 type of Runtime Exception in java

1) Unchecked Runtime Exception (Not required to listed in throws clause)
2) Checked Runtime Exception (Must be listed in throws clause)

Following are the some built in Unchecked Runtime Exception of java.

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- ClassCastException
- IllegalArgumentException
- IllegalMonitorStateException
- IllegalStateException
- IllegalThreadStateException
- IndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException
- NumberFormatException
- SecurityException
- StringIndexOutOfBounds
- UnsupportedOperationException

Following are the some built in Checked Runtime Exception of java.

- ClassNotFoundException
- CloneNotSupportedException
- IllegalAccessException
- InstantiationException
- InterruptedException
- NoSuchFieldException
- NoSuchMethodException

f) finally block

In java when any code line will generate the exception, after that line all code sequence is beaked and cursor direct moves to catch to handle that generated exception. To solve this problem we can use finally block.

Finally create a block of code that will be executed after a try or a catch block is completed.

The finally block execute whether or not an exception is thrown. If an exception is thrown, the finally block will execute even if no catch statement matches the exception. The finally clause is optional however each try statement requires at least one catch or a finally block.

Example – 12:abc.java

```
class abc
{
	public static void main(String s[])
	{
		try
		{
			System.out.println(10/0);
		}
		finally
		{
			System.out.println(" Hello ");
		}
	}
}
```

CHAPTER FIVE

Multithreading

IN THIS CHAPTER

- Multithreading

1) Multithreading

The java provides us to very use full tool that is multithreading. A multithreading programming work on the basis of multitasking operating system.

In multitasking operating system we will run more than one program/task at same time. For example we work with MS-word and listening songs in media player. Here MS word and media player are 2 different programs/tasks.

In java small block of code is known as a thread. The multithreading is work as same as multitasking. Multithreading allow us to execute more then one thread at the same time. The multithreading is depend on computer processor.

Let's take an example to understand multithreading mechanism.

- The processor gives us 10byte to execute our program.

10byte

- Now first thread of 4byte is go to in processor to execute.

10byte – 4byte **= 6byte**

- The second thread of 4byte also goes to in processor to execute.

10byte – 4byte – 4byte **= 2byte**

- Now the processor uses 8byte of available memory to execute these 2 threads. The third thread of 4byte is ready for execution and wait for availability of processor. Because in processor only 2byte memory is available but this thread wants 4byte to execute. When one of these 2 threads is over the third thread go in processor to execute.

- The fourth thread of 8byte is waiting for availability of processor, because now processor has only 2byte. Fourth thread will wait till the processor gives the memory of fourth thread wants to execute.

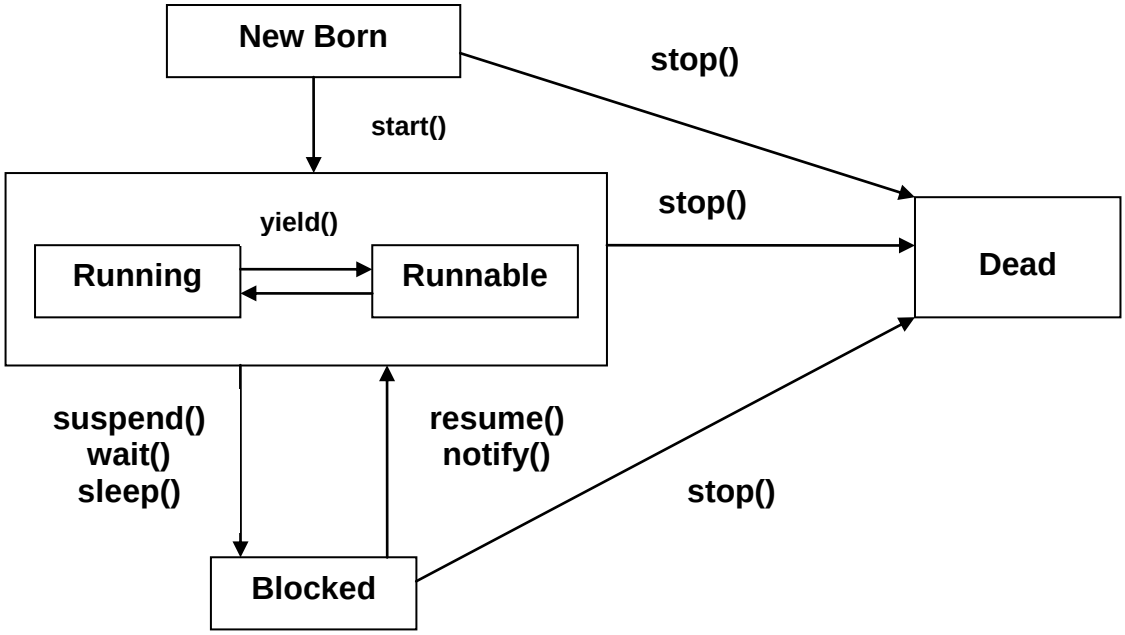
- Suppose one of these 2 threads is over. Now processor has 6byte memory. But fourth thread needs 8byte to execute.

2byte + 4byte **= 6byte**

- The processor gives this 6byte to other waiting threads in queue. In this queue any thread needs 6byte or less then 6byte to process, the processor give this memory to that thread but that thread comes first in waiting queue.

- In end the processor frees another 4byte. Now processor has 10byte memory and fourth thread is going in processor to execute.

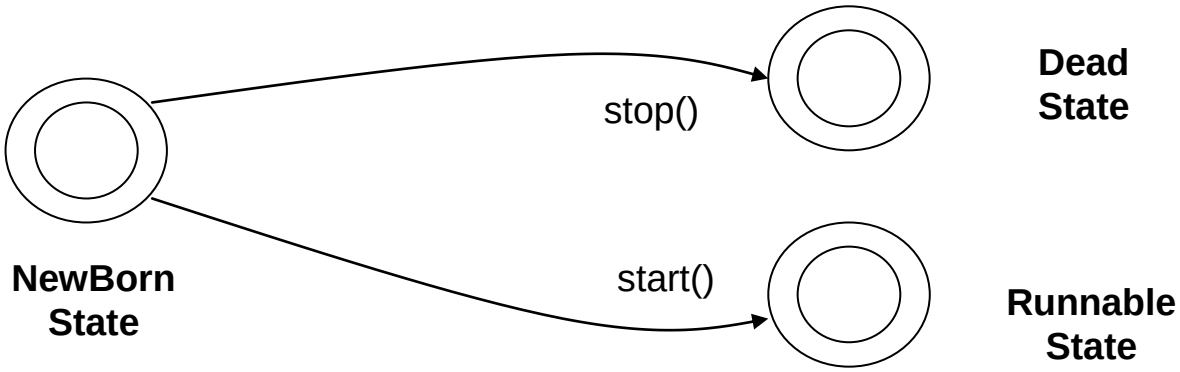
a) Thread Life Cycle



i) New born state

- When we create a thread object, the thread is born & is said to be in newborn state.

- At this state we can do only one of the following things with it.
  - Schedule it for running using start() method.
  - Kill it using stop() method.

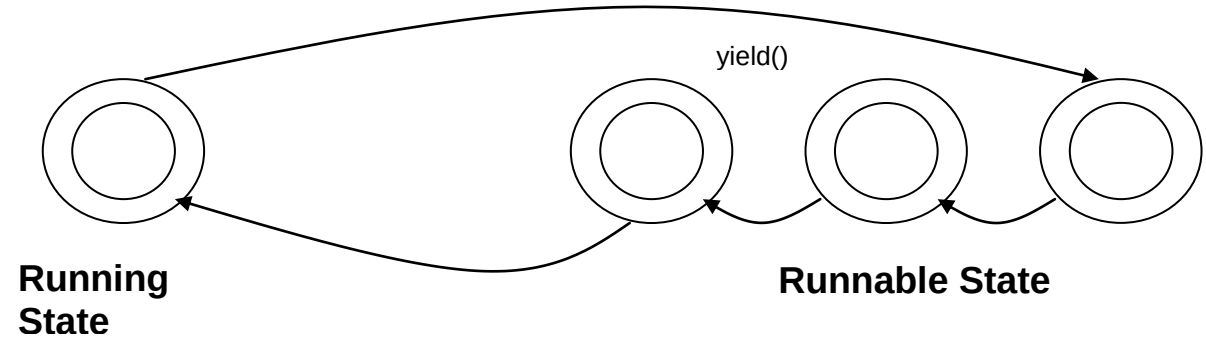


ii) Runnable state

- It means that the thread is ready for execution & it is waiting for the availability of the processor. The thread has joined the queue of threads that are waiting for execution.

- If all threads have equal priority, then they are given time slots for execution in round robin fashion. It means first come first serve manner. The process of assigning time to threads is known as **time slicing.**

- If we want to give control to another thread of equal priority before it turn comes, we can do so by using the yield() method.

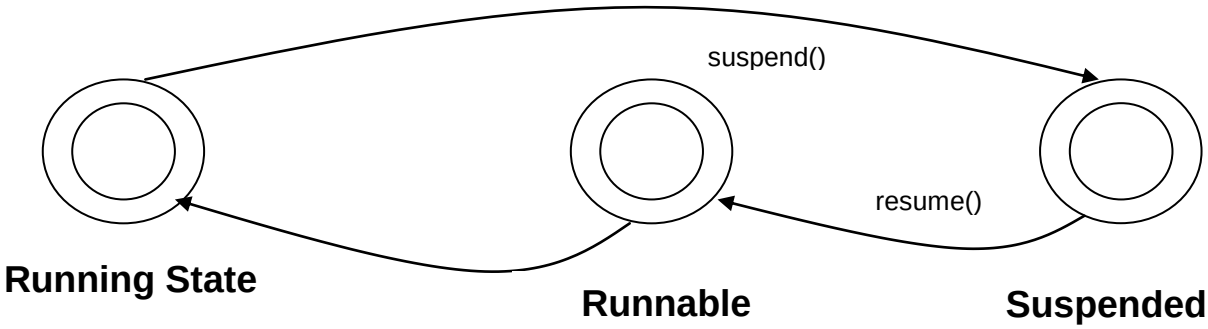


iii) Running State

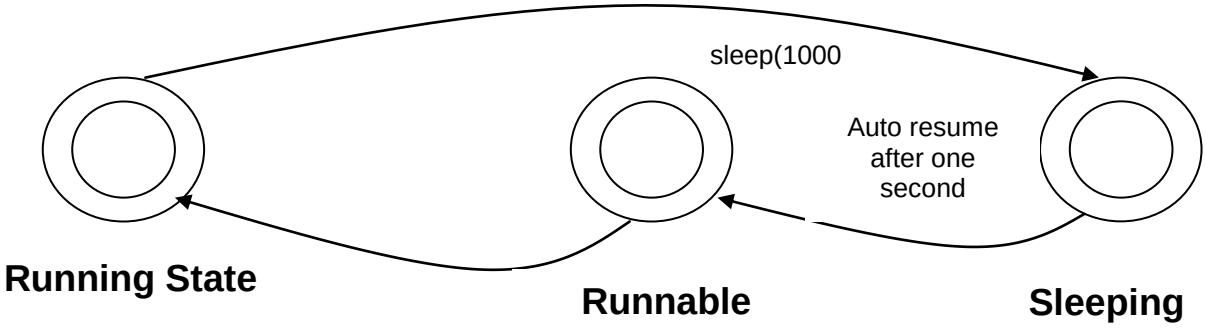
- It means that the processor has given its time to the thread for its execution.

- The thread runs until it gives control on its own or it is preempted by a higher priority thread. A running thread may give its control in one of the following situation.

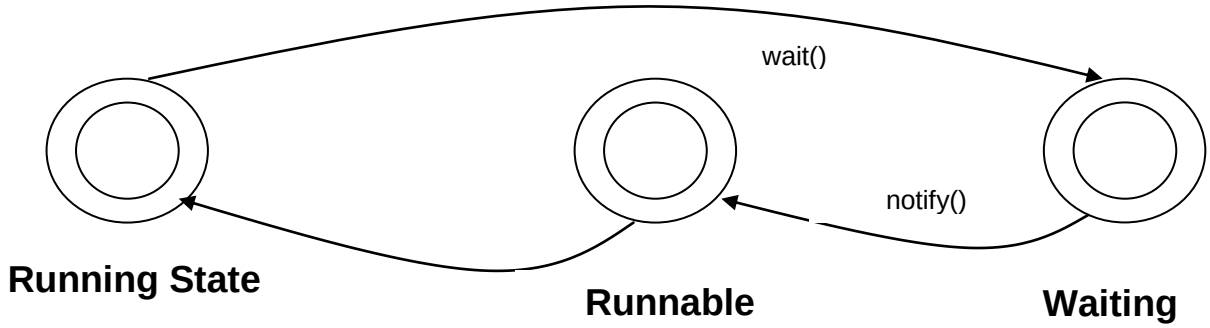
- It has been suspended using suspend() method a suspended thread can be resumed by using resume() method.



- It has been made to sleep. We can put the thread to sleep for a specified time period using the method sleep(time) where time is in milliseconds.



-It has been told to wait until some event occurs. This is done using wait() method. The tread can be scheduled to run again using the notify() method



iv) Blocked State

- A thread is said to be blocked when it is prevented from entering into the runnable state & subsequently the running state.

- This happens when the thread is suspended sleeping or waiting in order to satisfy certain requirements.

- A blocked thread is considered as "not runnable but not dead & therefore fully qualified to run again.

v) Dead State

- Every thread has a life cycle.

- A running thread ends its life when it has completed execution of it's run() method. it is a natural death of thread.

- We can kill it by sending the stop message using stop () to it any state thus causing a premature death to it.

- A thread can be killed as soon it is born or while it is running, or even when it is in not runnable condition.

b) Creating threads using Thread class or Runnable interface.

In java we can create thread using extending Thread class or implementing Runnable interface. Following are the Thread class constructors and methods are use to manages the threads.

i) public Thread()

- Creates a new Thread object. Automatically generated names are of the form "Thread-"+*n*, where *n* is an integer.

ii) public Thread(Runnable target)

- Creates a new Thread object. Automatically generated names are of the form "Thread-"+*n*, where *n* is an integer.
- **Parameters: target** - the object whose run method is called.

iii) public Thread(String name)

- Creates a new Thread object.
- **Parameters: name** - the name of the new thread.

iv) public Thread(Runnable target, String name)

- Creates a new Thread object.
- **Parameters: target** - the thread group.
  name - the name of the new thread.

v) public static Thread currentThread()

- Returns a reference to the currently executing thread object.
- **Returns:** The currently executing thread.

vi) public final String getName()

- Returns this thread's name.
- **Returns:** This thread's name.

vii)public final int getPriority()

- Returns this thread's priority.
- **Returns:** This thread's priority.

viii) public final boolean isAlive()

- Tests if this thread is alive. A thread is alive if it has been started and has not yet died.
- **Returns:** True if this thread is alive; false otherwise.

ix) public final void join() throws InterruptedException

- Waits for this thread to die.

x) public void run()

- If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called; otherwise, this method does

nothing and returns. Subclasses of Thread should override this method. It should never be called directly.

xi) public final void setName(String name)

- Changes the name of this thread to be equal to the argument name.
- **Parameters: name** - the new name for this thread.

xii)public final void setPriority(int newPriority)

- Changes the priority of this thread.
- **Parameters: newPriority** - priority to set this thread to

xiii) public static void sleep(long millis) throws InterruptedException

- Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors.
- **Parameters: millis** - the length of time to sleep in milliseconds.

xiv) public void start()

- Causes these thread to begin execution; the Java Virtual Machine calls the run method of this thread.
- When a java program starts up, one thread begins running immediately. This is usually calling the main thread of your program because it is the one that is executed when your program begins.
- A main thread is important for 2 reasons (1) it is the thread from which other child threads will be created (2) often it must be the last thread to finished the execution because it perform various shutdown actions.
- Although the main thread is created automatically when your program is started & it can be control by thread object. To do so you must obtain a reference to it.

Example – 1:abc.java

```
class abc
{
	public static void main(String s[])
	{
		Thread t = Thread.currentThread();
		System.out.println(t);
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(i);
```

```
                    Thread.sleep(1000);
                }
                catch(Exception e)
                {
                    System.out.println(e);
                }
            }
        }
}
```

c) Creating Thread using Runnable interface.

Example – 2:abc.java

```
class A implements Runnable
{
    Thread t;
    A()
    {
        t = new Thread(this, "Child Thread - 2");
        System.out.println(t);
        t.start();
    }
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            try
            {
                System.out.println(i);
                Thread.sleep(1000);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
class abc
{
    public static void main(String s[])
    {
```

```
                new A();
        }
}
```

Example – 3:abc.java

```
class A implements Runnable
{
        Thread t;
        A()
        {
                t = new Thread(this,"Child Thread - 1");
                System.out.println(t);
                t.start();
        }
        public void run()
        {
                for(int i=0;i<10;i++)
                {
                        try
                        {
                                System.out.println(t.getName()+" : "+i);
                                Thread.sleep(1000);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        }
}
class abc
{
        public static void main(String s[])
        {
                new A();
                Thread t = Thread.currentThread();
                t.setName(" Main Thread ");
                for(int i=0;i<10;i++)
                {
                        try
                        {
                                System.out.println(t.getName()+" : "+i);
```

```
                        Thread.sleep(1000);
                    }
                    catch(Exception e)
                    {
                        System.out.println(e);
                    }
                }
            }
}
```

Example – 4:abc.java

```
class A implements Runnable
{
        Thread t;
        A(int n)
        {
                t = new Thread(this,"Child Thread - "+n);
                System.out.println(t);
                t.start();
        }
        public void run()
        {
                for(int i=0;i<10;i++)
                {
                        try
                        {
                                System.out.println(t.getName()+" : "+i);
                                Thread.sleep(1000);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        }
}
class abc
{
        public static void main(String s[])
        {
                new A(1);
                new A(2);
```

```
		Thread t = Thread.currentThread();
		t.setName(" Main Thread ");
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(t.getName()+" : "+i);
				Thread.sleep(1000);
			}
			catch(Exception e)
			{
				System.out.println(e);
			}
		}
	}
}
```

Example – 5:abc.java

```
class A implements Runnable
{
	Thread t;
	A(int n)
	{
		t = new Thread(this,"A's Child Thread - "+n);
		System.out.println(t);
		t.start();
	}
	public void run()
	{
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(t.getName()+" : "+i);
				Thread.sleep(1000);
			}
			catch(Exception e)
			{
				System.out.println(e);
			}
		}
	}
```

```
}
class B implements Runnable
{
	Thread t;
	B(int n)
	{
		t = new Thread(this,"B's Child Thread - "+n);
		System.out.println(t);
		t.start();
	}
	public void run()
	{
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(t.getName()+" : "+i);
				Thread.sleep(1000);
			}
			catch(Exception e)
			{
				System.out.println(e);
			}
		}
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(1);
		new B(1);
		Thread t = Thread.currentThread();
		t.setName(" Main Thread ");
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(t.getName()+" : "+i);
				Thread.sleep(1000);
			}
			catch(Exception e)
```

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

d) Creating Thread using Thread class

Example – 6:abc.java

```
class A extends Thread
{
    A(int n)
    {
        super("Child thread - "+n);
        System.out.println(getName());
        start();
    }
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            try
            {
                System.out.println(i);
                Thread.sleep(1000);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
class abc
{
    public static void main(String s[])
    {
        new A(1);
    }
}
```

Example – 7:abc.java

```
class A extends Thread
{
	A(int n)
	{
		super("A's Child Thread - "+n);
		System.out.println(getName());
		start();
	}
	public void run()
	{
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(getName()+" : "+i);
				Thread.sleep(1000);
			}
			catch(Exception e)
			{
				System.out.println(e);
			}
		}
	}
}
class B extends Thread
{
	B(int n)
	{
		super("B's Child Thread - "+n);
		System.out.println(getName());
		start();
	}
	public void run()
	{
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(getName()+" : "+i);
				Thread.sleep(1000);
			}
```

```
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
    }
}
class abc
{
    public static void main(String s[])
    {
        new A(1);
        new B(1);
        Thread t = Thread.currentThread();
        t.setName(" Main Thread ");
        for(int i=0;i<10;i++)
        {
            try
            {
                System.out.println(t.getName()+" : "+i);
                Thread.sleep(1000);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
```

e) Use of isAlive() and join()

To see that one thread is ended or not, thread class provides method that check whether the thread is in running state or not.

2 ways exist to determine weather the thread has finish or not.
(1) You can call isAlive() on the thread this method define by thread class. It returns true if the thread still running otherwise it returns false.

final boolean isAlive()

(2) you can also call join() to wait for a thread to finished

final void join() throws InterrptedException

Example – 8:abc.java

```
class A implements Runnable
{
	Thread t;
	A(int n)
	{
		t = new Thread(this,"Child Thread - "+n);
		System.out.println(t);
		t.start();
	}
	public void run()
	{
		for(int i=0;i<10;i++)
		{
			try
			{
				System.out.println(t.getName()+" : "+i);
				Thread.sleep(1000);
			}
			catch(Exception e)
			{
				System.out.println(e);
			}
		}
	}
}
class abc
{
	public static void main(String s[])
	{
		A a = new A(1);
		A b = new A(2);
		Thread t = Thread.currentThread();
		t.setName(" Main Thread ");
		System.out.println(a.t.isAlive());
		System.out.println(b.t.isAlive());
		try
		{
			Thread.sleep(1000);
			b.t.join();
			a.t.join();
		}
```

```
            catch(Exception e)
            {
                System.out.println(e);
            }
            System.out.println(a.t.isAlive());
            System.out.println(b.t.isAlive());
        }
}
```

f) Thread priority

Java assigns to each thread a priority to determine how the thread should be treated with respect to other.

Thread priority is integer that specifies the relative priority of one thread to another. Instead a threads priority is used to decide when to switch from one running thread to the next. This is known as context switching.

Rules of context switching

(1) A thread can voluntary relinquish control. This is done by explicitly sleeping or blocking on pending input/output. In this all other thread are examine & the highest priority thread that is ready to run is given the CPU.

(2) A thread can be preempted by a higher priority thread. In this case the lower priority does not yield the processor is simply preempted. No meter what it is doing by a higher priority thread as soon as higher priority thread wants to run it does.

To set the thread priority use the setPrioriy() which is a member of Thread class

find void setpriority (int priority)

Here priority specifies the new priority for the calling thread

The value of priority must be within the range MIN_ PRIORITY & MAX_PRIORITY. The thread priority values are 1 & 10 respectively. To write a thread to a normal priority specifies NORM_PRIORITY which is currently 5.

Example – 9:abc.java

```
class A implements Runnable
{
        Thread t;
        A(int n,int p)
        {
            t = new Thread(this,"Child Thread - "+n);
            t.setPriority(p);
```

```
            System.out.println(t);
            t.start();
        }
        public void run()
        {
            for(int i=0;i<10;i++)
            {
                try
                {
                    System.out.println(t.getName()+" : "+i);
                    Thread.sleep(1000);
                }
                catch(Exception e)
                {
                    System.out.println(e);
                }
            }
        }
}
class abc
{
        public static void main(String s[])
        {
            new A(1,Thread.MAX_PRIORITY-3);
            new A(2,Thread.MIN_PRIORITY+2);
            Thread t = Thread.currentThread();
            t.setPriority(Thread.NORM_PRIORITY+1);
            t.setName(" Main Thread ");
            for(int i=0;i<10;i++)
            {
                try
                {
                    System.out.println(t.getName()+" : "+i);
                    Thread.sleep(1000);
                }
                catch(Exception e)
                {
                    System.out.println(e);
                }
            }
        }
}
```

g) Synchronization

As multithreading introduce on a synchronize behavior to your program, there must be a way to view enforce synchronicity when you need it.

For example if 2 thread communicate & share a complicated data structure such as a link list you need some way to ensure that they do not conflict with each other.

That is you must prevent to one thread from writing data while another thread is middle of reading it. For this purpose implements elegant on an age old model of inter process synchronization.

You can synchronize your code in 2 ways

(1) Using synchronized() method
(2) Using synchronized block

It means when two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

i) synchronized() method

To enter an object monitor just call a method that has been modify with the synchronized keyword

While the thread is inside synchronized() method all other threads that try to call it on the same instance have to wait.

To exit the monitor & relinquish (release) the control of the object to the next waiting thread the owner of the monitor simply return from the synchronized() method

Example – 10:abc.java

```
class A
{
	synchronized void disp(String nm)
	{
		try
		{
			for(int i=0;i<10;i++)
			{
				System.out.println(nm+":"+i);
				Thread.sleep(1000);
```

```
                }
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
}
class B implements Runnable
{
        Thread t;
        A a;
        B(String s1,A a1)
        {
            a=a1;
            t = new Thread(this,s1);
            System.out.println(t);
            t.start();
        }
        public void run()
        {
            a.disp(t.getName());
        }
}
class abc
{
        public static void main(String s[])
        {
            A a = new A();
            new B(" Thread - 1",a);
            new B(" Thread - 2",a);
        }
}
```

ii) synchronized block

The synchronized block also work same as synchronized() method. The main difference between both is that if we create synchronized() method than we must call it in the run() method.

Thus it is suitable to use synchronized block instead of synchronized() method.

Example – 11:abc.java

```
class A
{
	void disp(String nm)
	{
		try
		{
			for(int i=0;i<10;i++)
			{
				System.out.println(nm+":"+i);
				Thread.sleep(1000);
			}
		}
		catch(Exception e)
		{
			System.out.println(e);
		}
	}
}
class B implements Runnable
{
	Thread t;
	A a;
	B(String s1,A a1)
	{
		a=a1;
		t = new Thread(this,s1);
		System.out.println(t);
		t.start();
	}
	public void run()
	{
		synchronized(a)
		{
			a.disp(t.getName());
		}
	}
}
class abc
{
	public static void main(String s[])
	{
```

```
            A a = new A();
            new B(" Thread - 1",a);
            new B(" Thread - 2",a);
        }
}
```

h) Use of wait(), notify() and notifyAll()

wait() method tells the calling thread to give as the monitor & go to suspend until some other thread enter the same monitor & calls notify() method.

final void wait() throws InterruptedException

notify() method wakes up a threads that called wait() method on the same object**.**

final void notify()

notifyAll() method wakes up all the threads that called wait() method on the same object one of the threads will be granted access.

final void notifyAll()

wait() method normally waits until notify() or notifyAll() methods is called. This all 3 methods have been called only from within a synchronized context.

Example – 12:abc.java

```
class A
{
        boolean ok=false;
        synchronized void pause() throws Exception
        {
                while(!ok)
                {
                        wait();
                }
        }

        synchronized void start()
        {
                ok=true;
                notify();
        }
}
class B implements Runnable
{
```

```
        A a;
        Thread t;
        B(A a1)
        {
                a=a1;
                t = new Thread(this);
        t.start();
        }
        public void run()
        {
                try
                {
                        a.pause();
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
class abc
{
        public static void main(String[] args) throws Exception
        {
                Ath15 a = new Ath15();
                new Bth15(a);
                for (int i=0;i<10;i++)
                {
                        Thread.sleep(500);
                        System.out.print(".");
                }
                a.start();
        }
}
```

i) Deadlock

A special type of error that you need to avoid that relates specifically to multitasking is deadlock, which occurs when 2 threads have a circular dependency on a pair of synchronized object.

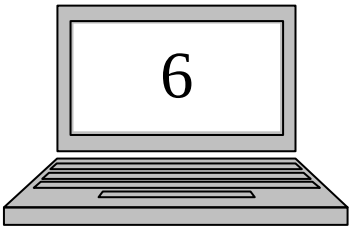
For example, suppose one thread enters the object monitor on object x & another thread enter the object monitor on object y. if thread x try to call any synchronized() method on y, it will block as excepted.

However the thread in y try to call any synchronized() for an thread waits forever. Because to access x it would have to release it own lock on y. so that the first thread should complete. It may involve more than 2 threads & 2 synchronize object.

Example – 13:abc.java

```
class A
{
	synchronized void x1(B b)
	{
		try
		{
			Thread.sleep(1000);
		}
		catch(InterruptedException e)
		{
			System.out.println (e);
		}
		b.y2();
	}
	synchronized void x2()
	{
		System.out.println ("hello");
	}
}
class B
{
	synchronized void y1(A a)
	{
		try
		{
			Thread.sleep(1000);
		}
		catch (InterruptedException e)
		{
			System.out.println (e);
		}
		a.x2();
	}
```

```
        synchronized void y2()
        {
                System.out.println ("world");
        }
}
class abc implements Runnable
{
        A a1 = new A();
        B b1 = new B();
        abc()
        {
                Thread t = new Thread(this);
                b1.y1(a1);
                t.start();
        }
        public void run()
        {
                a1.x1(b1);
        }
        public static void main(String s[])
        {
                new abcth16();
        }
}
```



CHAPTER SIX

JAVA Packages

IN THIS CHAPTER

- Java Lang Packages
- Java.Util Packages
- Java.io.Package

1) java.lang package

a) Object class

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All types of objects, including arrays, implement the methods of this class.

i) public boolean equals(Object obj)

- Indicates whether invoking object is "equal to" specified object. If both objects are equal than this method return true otherwise false. This method is case sensitive.

Example – 1:abc.java

```
class abc
{
	public static void main(String s[])
	{
		Object o1 = " Hello ";
		Object o2 = 10;
		if(o1.equals(o2))
		{
			System.out.println("ok");
		}
		else
		{
			System.out.println("no");
		}
	}
}
```

ii) protected void finalize()throws Throwable

- Called by the garbage collector on an object when garbage collection determines that there are no more references to the object. A subclass overrides the finalize method to dispose of system resources or to perform other cleanup.

iii) public final void notify()

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

iv) public final void notifyAll()

- Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods.

v) public final void wait() throws InterruptedException

- Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

b) Math class

The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

This Class has 2 constant.

- **E :** Exponential constant value approx 2.718.
- **PI :** approx value 3.14

i) public static double abs(int a)

- Returns the absolute value of a double value. If the argument is not negative, the argument is returned.

ii) public static double ceil(double a)

- Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.

iii) public static double floor(double a)

- Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.

iv) public static double sin(double a)

- Returns the trigonometric sine of an angle.

v) public static double cos(double a)

- Returns the trigonometric cosine of an angle.

vi) public static double tan(double a)

- Returns the trigonometric tangent of an angle.

vii)public static double exp(double a)

- Returns Euler's number *e* raised to the power of a double value.

viii) public static double log(double a)

- Returns the natural logarithm (base *e*) of a double value.

ix) public static double log10(double a)

- Returns the base 10 logarithm of a double value.

x) public static double sqrt(double a)

- Returns the correctly rounded positive square root of a double value.

xi) public static double pow(double a,double b)

- Returns the value of the first argument raised to the power of the second argument.

xii) public static int round(double a)

- Returns the closest int to the argument.

xiii) public static int max(int a,int b)

- Returns the greater of two int values.

xiv) public static int min(int a,int b)

- Returns the smaller of two int values.

xv) public static double signum(double d)

- Returns zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.

Example:abc.java

```
class abc
{
	public static void main(String s[])
	{
		System.out.println("abs(-100) : "+Math.abs(-100));
		System.out.println("abc(100) : "+Math.abs(100));
		System.out.println("ceil(5.88) : "+Math.ceil(5.88));
		System.out.println("floor(5.22) : "+Math.floor(5.22));
		System.out.println("sin(12) : "+Math.sin(12));
		System.out.println("cos(12) : "+Math.cos(12));
		System.out.println("tan(12) : "+Math.tan(12));
		System.out.println("exp(2) : "+Math.exp(2));
		System.out.println("log(2) : "+Math.log(2));
		System.out.println("log10(2) : "+Math.log10(2));
		System.out.println("sqrt(4) : "+Math.sqrt(4));
```

```
        System.out.println("pow(2,5) : "+Math.pow(2,5));
        System.out.println("round(10.521) : "+Math.round(10.521));
        System.out.println("round(-10.521) : "+Math.round(-10.521));
        System.out.println("max(10,14) : "+Math.max(10,14));
        System.out.println("min(10,14) : "+Math.min(10,14));
        System.out.println("signum(-10) : "+Math.signum(-10));
        System.out.println("signum(10) : "+Math.signum(10));
        System.out.println("signum(0) : "+Math.signum(0));
    }
}
```

c) String class

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared.

To concate more than one String literal or object use '+' operator.

For example,

```
System.out.println(" Hello "+" World");      // Hello World
System.out.println(" Hello "+2+2);           // Hello22
System.out.println(" Hello "+(2+2));         // Hello4
```

i) String()

- To create an empty String.

ii) String(char ch[])

- To create a string initialize by an array of character used following constructor.

Example – 1:abc.java

```
class abc
{
    public static void main(String s[])
    {
        char c[]={'a','b','c','d'};
        String s1 = new String(c);
        System.out.println(s1);              //abcd
    }
}
```

iii) String(char ch[],int start,int no_of_char)

- To create a sub-range of specified character array.

Example – 2:abc.java

```
class abc
{
	public static void main(String s[])
	{
		char c[]={'a','b','c','d','e','f',g'};
		String s1 = new String(c,2,2);
		System.out.println(s1);			//cd
	}
}
```

iv) String(String str)

- To create a String object from another String object.

Example – 3:abc.java

```
class abc
{
	public static void main(String s[])
	{
		char c[]={'a','b','c','d'};
		String s1 = new String(c);
		String s2 = new String(s1);
		System.out.println(s2);			//abcd
		System.out.println(s1);			//abcd
	}
}
```

v) String(byte b[])

- To convert bytes into its related ASCII characters.

Example – 4:abc.java

```
class abc
{
	public static void main(String s[])
	{
		byte b[]={65,97,66,98,67,99};
		String s1 = new String(b);
		System.out.println(s1);			//AaBbCc
	}
}
```

vi) public int length()

- TO find invoking String length.

Example – 5:abc.java

```
Class abc
{
	Public static void main(String s[])
	{
		System.out.println(" Length of Hello : "+hello.length());   //5
		/*
			String s1 = new String("Hello");
			System.out.println((" Length of Hello : "+s1.length());      //5
		*/
	}
}
```

vii)public char charAt(int index)

- Returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

viii) public String concat(String str)

- Concatenates the specified string to the end of this string.

ix) public boolean endsWith(String suffix)

- Tests if this string ends with the specified suffix.

x) public boolean equals(Object anObject)

- Compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.

xi) public boolean equalsIgnoreCase(String anotherString)

- Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length and corresponding characters in the two strings are equal ignoring case.

xii)public byte[] getBytes()

- Convert this String into ASCII value and storing the result into a new byte array.

xiii) public void getChars(int srcBegin,int srcEnd,char[] dst,int dstBegin)

- Copies characters from this string into the destination character array.

Here, **srcBegin** - index of the first character in the string to copy.
srcEnd - index after the last character in the string to copy.
dst - the destination array.
dstBegin - the start offset in the destination array.

xiv) public int indexOf(int ch / char ch)

- Returns the index within this string of the first occurrence of the specified character.

xv) public int indexOf(int ch / char ch,int fromIndex)

- Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

xvi) public int indexOf(String str)

- Returns the index within this string of the first occurrence of the specified substring

xvii) public int indexOf(String str,int fromIndex)

- Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

xviii) public boolean isEmpty()

- Returns true if, and only if, length() is 0.

xix) public int lastIndexOf(int ch)

- Returns the index within this string of the last occurrence of the specified character.

xx) public int lastIndexOf(int ch,int fromIndex)

- Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

xxi) public int lastIndexOf(String str)

- Returns the index within this string of the rightmost occurrence of the specified substring.

xxii) public int lastIndexOf(String str,int fromIndex)

- Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

xxiii) public boolean regionMatches(int toffset,String other,int ooffset,int len)

- Tests if two string regions are equal.

- Here, **toffset** - the starting offset of the subregion in this string.
  other - the string argument.
  ooffset - the starting offset of the subregion in the string argument.
  len - the number of characters to compare.

xxiv) public String replace(char oldChar,char newChar)
- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

xxv) public String replace(CharSequence target,CharSequence replacement)
- Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.

xxvi) public boolean startsWith(String prefix)
- Tests if this string starts with the specified prefix.

xxvii) public String substring(int beginIndex)
- Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

xxviii) public String substring(int beginIndex,int endIndex)
- Returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex.

xxix) public char[] toCharArray()
- Converts this string to a new character array.

xxx) public String toLowerCase()
- Converts all of the characters in this String to lower case.

xxxi) public String toUpperCase()
- Converts all of the characters in this String to upper case

xxxii) public String trim()
- Returns a copy of the string, with leading and trailing whitespace omitted.

xxxiii) public byte[] getBytes()
- Returns byte array of the String.

Example – 6:abc.java

```
class abc
{
	public static void main(String s[])
	{
		String s1 = " fp ";
		String s2 = "abc_opq_xyz_mno_opq";
		System.out.println(" charAt() : "+s2.charAt(4));	//o
System.out.println(" concat() : "+s1.concat(s2));	// fp abc_opq_xyz_mno
System.out.println(" startsWith() : "+s2.startsWith("a")); //true
System.out.println(" startsWith() : "+s2.startsWith("abc"));//true
System.out.println(" endsWith() : "+s2.endsWith("q"));	//true
System.out.println(" endsWith() : "+s2.endsWith("opq"));		//true
		System.out.println(" equals() : "+s1.equals("Hello"));	//false
		System.out.println(" equals() : "+s1.equals(" fp ")); //true
System.out.println(" equalsIgnoreCase() : "+s1.equalsIgnoreCase("Hello"));	//false
System.out.println(" equalsIgnoreCase() : "+s1.equalsIgnoreCase(" FP "));	//true
		System.out.println(" getBytes() ");
		byte b[] = s2.getBytes();
		for(int i=0;i<b.length;i++)
		{
			System.out.println(b[i]);
		}
		System.out.println(" getChars() ");
		char c[] = new char[10];
		s2.getChars(1,10,c,0);
		for(int i=0;i<c.length;i++)
		{
			System.out.println(c[i]);
		}
		System.out.println(" indexOf() : "+s2.indexOf('o')); //4
		System.out.println(" indexOf() : "+s2.indexOf(97)); //0
		System.out.println(" indexOf() : "+s2.indexOf('o',5));	//14
		System.out.println(" indexOf() : "+s2.indexOf(97,5));	//-1
		System.out.println(" indexOf() : "+s2.indexOf("opq"));	//4
		System.out.println(" indexOf() : "+s2.indexOf("opq",5));//16
System.out.println(" isEmpty() : "+new String().isEmpty());		//true
System.out.println(" isEmpty() : "+new String("Hello").isEmpty());	//false
		System.out.println(" lastIndexOf() : "+s2.lastIndexOf('o'));		//16
		System.out.println(" lastIndexOf() : "+s2.lastIndexOf(97));		//0
	System.out.println(" lastIndexOf() : "+s2.lastIndexOf('o',5));	//4
	System.out.println(" lastIndexOf() : "+s2.lastIndexOf(97,5));	//0
```

```
        System.out.println(" lastIndexOf() : "+s2.lastIndexOf("opq"));    //16
        System.out.println(" lastIndexOf() : "+s2.lastIndexOf("opq",5));//4
        System.out.println("                    regionMatches()                    :
        "+s2.regionMatches(0,"abc_opq_xyz",0,10));//true
System.out.println(" replace() : "+s2.replace('_','-'));        //abc-opq-xyz-mno-opq
System.out.println(" replace() : "+s2.replace("opq","asd"));//abc-asd-xyz-mno-asd
System.out.println(" substring() : "+s2.substring(10));//z_mno_opq
System.out.println(" substring() : "+s2.substring(10,17)); //z_mno_o
                System.out.println(" toCharArray() ");
                char ch[]=s2.toCharArray();
                for(int i=0;i<ch.length;i++)
                {
                        System.out.println(ch[i]);
                }
System.out.println(" toLowerCase() : "+new String("HELLO").toLowerCase());//hello
        System.out.println(" toUpperCase() : "+s1.toUpperCase());        // FP
        System.out.println(" trim() : "+s1.trim());
        }
}
```

d) StringBuffer class

It is a mutable sequence of characters. A string buffer is like a String, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

StringBuffer will automatically grows to make room to such editions & often has more character pre-allocted that are actually need to allow room for growth

i) public StringBuffer()

- Creates a string buffer with no characters in it and an initial capacity of 16 characters.

ii) public StringBuffer(int capacity)

- Creates a string buffer with no characters in it and the specified initial capacity.

iii) public StringBuffer(String str)

- Creates a string buffer initialized to the contents of the specified string. The initial capacity of the string buffer is 16 plus the length of the string argument.

iv) public StringBuffer append(String str)

- Appends the specified string to this character sequence. This methods changes into the original one.

v) public int capacity()

- Returns the current capacity. The capacity is the amount of storage available for newly inserted characters, beyond which an allocation will occur.

vi) public char charAt(int index)

- Returns the char value in this sequence at the specified index. The first char value is at index 0, the next at index 1, and so on, as in array indexing.
- The index argument must be greater than or equal to 0, and less than the length of this sequence.

vii)public StringBuffer delete(int start,int end)

- Removes the characters in a substring of this sequence. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists. If start is equal to end, no changes are made.

viii) public StringBuffer deleteCharAt(int index)

- Removes the char at the specified position in this sequence. This sequence is shortened by one char.

ix) public void ensureCapacity(int minimumCapacity)

- Ensures that the capacity is at least equal to the specified minimum. If the current capacity is less than the argument, then a new internal array is allocated with greater capacity. The new capacity is the larger of:
- The minimumCapacity argument.
  Twice the old capacity, plus 2.

x) public StringBuffer insert(int offset,String str)

- Inserts the string into this character sequence.

xi) public StringBuffer reverse()

- Causes this character sequence to be replaced by the reverse of the sequence.

xii)public void setCharAt(int index,char ch)

- The character at the specified index is set to ch. This sequence is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character ch at position index. The

index argument must be greater than or equal to 0, and less than the length of this sequence.

xiii) public void trimToSize()

- Attempts to reduce storage used for the character sequence. If the buffer is larger than necessary to hold its current sequence of characters, then it may be resized to become more space efficient.

Example :abc.java

```
class abc
{
	public static void main(String s[])
	{
		StringBuffer sb1 = new StringBuffer();
		System.out.println(" capacity() : "+sb1.capacity()); //16
		StringBuffer sb2 = new StringBuffer(20);
		System.out.println(" capacity() : "+sb2.capacity()); //20
		StringBuffer sb3 = new StringBuffer(" Hello ");
		System.out.println(" capacity() : "+sb3.capacity()); //23
		System.out.println(" append() ");
		sb3.append("fp How are you ?");
		System.out.println(sb3);  // Hello fp How are you ?
		System.out.println(" charAt() : "+sb3.charAt(3));     //l
	System.out.println(" delete() : "+sb3.delete(11,23));        // Hello fp H
System.out.println(" deleteCharAt() : "+sb3.deleteCharAt(10));// Hello fp
		sb1.ensureCapacity(20);
		System.out.println(" capacity() : "+sb1.capacity()); //34
System.out.println("insert():"+sb3.insert(6,"How are you ?"));// Hello How are you  ? fp
		System.out.println(" setCharAt() ");
		sb3.setCharAt(6,',');
		System.out.println(sb3);  // Hello,How are you ? fp
	}
}
```

e) Wrapper classes

Java use primitive types, such as int & char. This data types are not part of the object hierarchy. They are passed by value to methods & can to be directly passed by reference. Also, there is no way for 2 methods to refer to the same instance of an int
At times you will need to create an object representation of one of these primitive types for example, there are collection classes that deals only with objects, to store a

primitive type in one of these wrapper classes, you need to wrap the primitive type in a class

Java provides classes that correspond to each of the primitive types in essence; these classes encapsulate or wrap the primitive types within a class.

f) Number classes

The abstract class number defines a super class that is implemented by the classes that wrap the numeric types like byte, short, int, long, float & double

Number class has abstract methods that return the value of the object in each of the different formats.

for example doubleValue () returns the value as Double; floatValue () returns Float & so on.

byte byteValue ()
double doubleValue()
float floatVlalue ()
int intValue ()
long longValue()
short shortValue ()

g) Byte class

i) public Byte(byte value)
- Constructs a newly allocated Byte object that represents the specified byte value.

ii) public Byte(String s)throws NumberFormatException
- Constructs a newly allocated Byte object that represents the byte value indicated by the String parameter.

iii) public static byte parseByte(String s)throws NumberFormatException
- Parses the string argument as a signed decimal byte.

iv) public String toString()
- Returns a String object representing this Byte's value.

v) public boolean equals(Object obj)

- Compares this object to the specified object. The result is true if and only if the argument is not null and is a Byte object that contains the same byte value as this object.

vi) public int compareTo(Byte anotherByte)

- Compares two Byte objects numerically.
- **Parameters: anotherByte** - the Byte to be compared.
- **Returns:** The value 0 if this Byte is equal to the argument Byte; a value less than 0 if this Byte is numerically less than the argument Byte; and a value greater than 0 if this Byte is numerically greater than the argument.

h) Short class

i) public Short(short value)

- Constructs a newly allocated Short object that represents the specified short value.

ii) public Short(String s)throws NumberFormatException

- Constructs a newly allocated Short object that represents the short value indicated by the String parameter.

iii) public String toString()

- Returns a String object representing this Short's value.

iv) public static short parseShort(String s) throws NumberFormatException

- Parses the string argument as a signed decimal short.

v) public boolean equals(Object obj)

- Compares this object to the specified object. The result is true if and only if the argument is not null and is a Short object that contains the same short value as this object.

vi) public int compareTo(Short anotherShort)

- Compares two Short objects numerically.
- **Parameters: anotherShort** - the Short to be compared.
- **Returns:** The value 0 if this Short is equal to the argument Short; a value less than 0 if this Short is numerically less than the argument Short; and a value greater than 0 if this Short is numerically greater than the argument Short.

i) Integer class

i) public Integer(int value)

- Constructs a newly allocated Integer object that represents the specified int value.

ii) public Integer(String s) throws NumberFormatException

- Constructs a newly allocated Integer object that represents the int value indicated by the String parameter.

iii) public static int parseInt(String s) throws NumberFormatException

- Parses the string argument as a signed decimal integer.

iv) public String toString()

- Returns a String object representing this Integer's value.

v) public boolean equals(Object obj)

- Compares this object to the specified object. The result is true if and only if the argument is not null and is an Integer object that contains the same int value as this object.

vi) public int compareTo(Integer anotherInteger)

- Compares two Integer objects numerically.
- **Parameters: anotherInteger** - the Integer to be compared.
- **Returns:** the value 0 if this Integer is equal to the argument Integer; a value less than 0 if this Integer is numerically less than the argument Integer; and a value greater than 0 if this Integer is numerically greater than the argument Integer.

vii)public static int signum(int i)

- Returns the signum function of the specified int value. The return value is -1 if the specified value is negative; 0 if the specified value is zero; and 1 if the specified value is positive.

j) Long class

i) public Long(long value)

- Constructs a newly allocated Long object that represents the specified long argument.

ii) public Long(String s) throws NumberFormatException

- Constructs a newly allocated Long object that represents the long value indicated by the String parameter.

iii) public static long parseLong(String s)throws NumberFormatException

- Parses the string argument as a signed decimal long.

iv) public String toString()

- Returns a String object representing this Long's value.

v) public boolean equals(Object obj)

- Compares this object to the specified object. The result is true if and only if the argument is not null and is a Long object that contains the same long value as this object.

vi) public int compareTo(Long anotherLong)

- Compares two Long objects numerically.
- **Parameters: anotherLong** - the Long to be compared.
- **Returns:** The value 0 if this Long is equal to the argument Long; a value less than 0 if this Long is numerically less than the argument Long; and a value greater than 0 if this Long is numerically greater than the argument Long.

vii)public static int signum(long i)

- Returns the signum function of the specified long value. (The return value is -1 if the specified value is negative; 0 if the specified value is zero; and 1 if the specified value is positive.)

k) Float class

i) public Float(float value)

- Constructs a newly allocated Float object that represents the primitive float argument.

ii) public Float(double value)

- Constructs a newly allocated Float object that represents the argument converted to type float.

iii) public Float(String s)throws NumberFormatException

- Constructs a newly allocated Float object that represents the floating-point value of type float represented by the string.

iv) public static float parseFloat(String s)throws NumberFormatException

- Returns a new float initialized to the value represented by the specified String.

v) public static boolean isNaN(float v)

- Returns true if the specified number is a Not-a-Number (NaN) value, false otherwise.

vi) public boolean isNaN()

- Returns true if this Float value is a Not-a-Number (NaN), false otherwise.

vii)public String toString()

- Returns a string representation of this Float object.

viii) public boolean equals(Object obj)

- Compares this object against the specified object.

ix) public int compareTo(Float anotherFloat)

- Compares two Float objects numerically.
- **Returns:** The value 0 if anotherFloat is numerically equal to this Float; a value less than 0 if this Float is numerically less than anotherFloat; and a value greater than 0 if this Float is numerically greater than anotherFloat.

x) public static int compare(float f1,float f2)

- Compares the two specified float values.
- **Returns:** The value 0 if f1 is numerically equal to f2; a value less than 0 if f1 is numerically less than f2; and a value greater than 0 if f1 is numerically greater than f2.

l) Double class

i) public Double(double value)

- Constructs a newly allocated Double object that represents the primitive double argument.

ii) public Double(String s)throws NumberFormatException

- Constructs a newly allocated Double object that represents the floating-point value of type double represented by the string.

iii) public static double parseDouble(String s)throws NumberFormatException

- Returns a new double initialized to the value represented by the specified String

iv) public static boolean isNaN(double v)

- Returns true if the specified number is a Not-a-Number (NaN) value, false otherwise.

v) public boolean isNaN()

- Returns true if this Double value is a Not-a-Number (NaN), false otherwise.

vi) public String toString()

- Returns a string representation of this Double object.

vii)public boolean equals(Object obj)

- Compares this object against the specified object.

viii) public int compareTo(Double anotherDouble)

- Compares two Double objects numerically.
- **Returns:** The value 0 if anotherDouble is numerically equal to this Double; a value less than 0 if this Double is numerically less than anotherDouble; and a value greater than 0 if this Double is numerically greater than anotherDouble.

ix) public static int compare(double d1,double d2)

- Compares the two specified double values.
- **Returns:** The value 0 if d1 is numerically equal to d2; a value less than 0 if d1 is numerically less than d2; and a value greater than 0 if d1 is numerically greater than d2.

m)Character class

i) public Character(char value)

- Constructs a newly allocated Character object that represents the specified char value.

ii) public char charValue()

- Returns the value of this Character object.

iii) public boolean equals(Object obj)

- Compares this object against the specified object.

iv) public String toString()

- Returns a String object representing this Character's value.

v) public static String toString(char c)

- Returns a String object representing the specified char.

vi) public static boolean isLowerCase(char ch)

- Determines if the specified character is a lowercase character.

vii)public static boolean isUpperCase(char ch)

- Determines if the specified character is an uppercase character.

viii) public static boolean isDigit(char ch)

- Determines if the specified character is a digit.

ix) public static boolean isLetter(char ch)

- Determines if the specified character is a letter.

x) public static boolean isLetterOrDigit(char ch)

- Determines if the specified character is a letter or digit.

xi) public static char toLowerCase(char ch)

- Converts the character argument to lowercase.

xii)public static char toUpperCase(char ch)

- Converts the character argument to uppercase.

xiii) public static boolean isWhitespace(char ch)

- Determines if the specified character is white space according to Java.

n) Boolean class

i) public Boolean(boolean value)

- Allocates a Boolean object representing the value argument.

ii) public Boolean(String s)

- Allocates a Boolean object representing the value true if the string argument is not null and is equal, ignoring case, to the string "true". Otherwise, allocate a Boolean object representing the value false.
- **Examples:**
  new Boolean("True") produces a Boolean object that represents true.
  new Boolean("yes") produces a Boolean object that represents false.

iii) public static boolean parseBoolean(String s)

- Parses the string argument as a boolean. The boolean returned represents the value true if the string argument is not null and is equal, ignoring case, to the string "true".

- **Example:**
  Boolean.parseBoolean("True")returns true.
  Boolean.parseBoolean("yes") returns false.

iv) public boolean booleanValue()
- Returns the value of this Boolean object as a boolean primitive.

v) public static String toString(boolean b)
- Returns a String object representing the specified boolean. If the specified boolean is true, then the string "true" will be returned, otherwise the string "false" will be returned.

vi) public String toString()
- Returns a String object representing this Boolean's value. If this object represents the value true, a string equal to "true" is returned. Otherwise, a string equal to "false" is returned.

2)java.util package

This package contains a large assortment of classes & interfaces that support a wide range of functionality.

For example, java.util has classes that generate pseudorandom numbers, manage data & time observer events, manipulate set of bits, tokenize string & handle formatted data.

This package also contains one of java's most powerful sub-system that is The collection framework.The collection framework is a sophisticated hierarchy of interfaces & classes that provide state of the technology for managing groups of object.

a) Enumeration interface

An object that implements the Enumeration interface generates a series of elements, one at a time.

i) public interface Enumeration<E>
- Where E specifies the types of element being enumerated.

ii) boolean hasMoreElements()
- Tests if this enumeration contains more elements.

- **Returns:** true if and only if this enumeration object contains at least one more element to provide; false otherwise.

iii) E nextElement()

- Returns the next element of this enumeration if this enumeration object has at least one more element to provide.
- **Returns:** the next element of this enumeration.

b) Vector class

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

Class vector <E>

Where E specifies the type of element that will be stored

This class has 3 data members.

- **protected Object[] elementData**

The array buffer into which the components of the vector are stored. The capacity of the vector is the length of this array buffer, and is at least large enough to contain all the vector's elements. Any array elements following the last element in the Vector are null.

- **protected int elementCount**

The number of valid components in this Vector object. Components elementData[0] through elementData[elementCount-1] are the actual items.

- **protected int capacityIncrement**

The amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity. If the capacity increment is less than or equal to zero, the capacity of the vector is doubled each time it needs to grow.

i) public Vector()

- Constructs an empty vector so that its internal data array has size 10 and its standard capacity increment is zero.

ii) public Vector(int initialCapacity)

- Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
- **Parameters: initialCapacity** - the initial capacity of the vector

iii) public Vector(int initialCapacity, int capacityIncrement)

- Constructs an empty vector with the specified initial capacity and capacity increment.
- **Parameters: initialCapacity** - the initial capacity of the vector
  capacityIncrement - the amount by which the capacity is increased when the vector overflows

iv) public void addElement(E obj)

- Adds the specified component to the end of this vector, increasing its size by one. The capacity of this vector is increased if its size becomes greater than its capacity.

v) public int capacity()

- Returns the current capacity of this vector.
- **Returns:** the current capacity

vi) public void clear()

- Removes all of the elements from this Vector. The Vector will be empty after this call returns

vii)public boolean contains(Object o)

- Returns true if this vector contains the specified element.

viii) public Enumeration<E> elements()

- Returns an enumeration of the components of this vector. The returned Enumeration object will generate all items in this vector. The first item generated is the item at index 0, then the item at index 1, and so on.
- **Returns:** an enumeration of the components of this vector

ix) public boolean isEmpty()

- Tests if this vector has no components.

x) public boolean removeElement(Object obj)

- Removes the first (lowest-indexed) occurrence of the argument from this vector. If the object is found in this vector, each component in the vector with an index greater or equal to the object's index is shifted downward to have an index one smaller than the value it had previously.

xi) public void setSize(int newSize)

- Sets the size of this vector. If the new size is greater than the current size, new null items are added to the end of the vector. If the new size is less than the current size, all components at index newSize and greater are discarded.

xii)public int size()

- Returns the number of components in this vector.

xiii) public void setElementAt(E obj,int index)

- Sets the component at the specified index of this vector to be the specified object. The previous component at that position is discarded. The index must be a value greater than or equal to 0 and less than the current size of the vector.

xiv) public void add(int index,E element)

- Inserts the specified element at the specified position in this Vector. Shifts the element currently at that position to the right (adds one to their indices).

Example :abc.java

```
import java.util.*;
class abc
{
	public static void main(String s[])
	{
		Vector<Integer> v1 = new Vector<Integer>();
		Vector<Integer> v2 = new Vector<Integer>(20);
		Vector<Integer> v3 = new Vector<Integer>(3,2);
		System.out.println(" capacity() : "+v1.capacity());   //10
		System.out.println(" capacity() : "+v2.capacity());   //20
		System.out.println(" capacity() : "+v3.capacity());   //3
		v3.addElement(10);
		v3.addElement(15);
		v3.addElement(20);
		System.out.println(" capacity() : "+v3.capacity());   //5
		System.out.println(" contains() : "+v3.contains(10));       //true
		System.out.println(" size() : "+v1.size());                 //0
		System.out.println(" size() : "+v2.size());                 //0
		System.out.println(" size() : "+v3.size());                 //4
		System.out.println(" isEmpty() : "+v1.isEmpty());    //true
		System.out.println(" isEmpty() : "+v2.isEmpty());    //true
		System.out.println(" isEmpty() : "+v3.isEmpty());    //false
	System.out.println(" removeElement() : "+v3.removeElement(10));     //true
		Enumeration<Integer> e1 = v3.elements();
		while(e1.hasMoreElements())
		{
			System.out.println(e1.nextElement());
		}
```

```
            v3.add(0,9);
            e1 = v3.elements();
            System.out.println(" After element is added ");
            while(e1.hasMoreElements())
            {
                  System.out.println(e1.nextElement());
            }
            v3.setElementAt(10,0);
            e1 = v3.elements();
            System.out.println(" After element is replaced ");
            while(e1.hasMoreElements())
            {
                  System.out.println(e1.nextElement());
            }
            System.out.println(" isEmpty() : "+v3.isEmpty());    //false
            v3.clear();
            System.out.println(" isEmpty() : "+v3.isEmpty());    //true
      }
}
```

c) Stack class

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack.

i) public class Stack<E> extends Vector<E>

- Where E specifies the type of element that will be stored

ii) public Stack()

- Creates an empty Stack.

iii) public E push(E item)

- Pushes an item onto the top of this stack.

iv) public E pop()

- Removes the object at the top of this stack and returns that object as the value of this function.
- **Returns:** The object at the top of this stack.

v) public E peek()

- Looks at the object at the top of this stack without removing it from the stack.

- **Returns:** the object at the top of this stack.

vi) public boolean empty()
- Tests if this stack is empty.
- **Returns:** true if and only if this stack contains no items; false otherwise.

vii)public int search(Object o)
- Returns the 1-based position where an object is on this stack. If the object o occurs as an item in this stack, this method returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance 1.
- **Parameters: o** - the desired object.
- **Returns:** The 1-based position from the top of the stack where the object is located; the return value -1 indicates that the object is not on the stack.

Example : abc.java

```
import java.util.*;
class abc
{
	public static void main(String s[])
	{
		Stack<String> s1 = new Stack<String>();
		s1.push("fp");
		s1.push("abc");
		s1.push("xyz");
		System.out.println(" empty() : "+s1.empty());		//false
		System.out.println(" search() : "+s1.search("xyz"));//1
		System.out.println(" peek() : "+s1.peek());		//xyz
		System.out.println(" pop() : "+s1.pop());		//xyz
		System.out.println(" peek() : "+s1.peek());		//abc
		System.out.println(" pop() : "+s1.pop());		//abc
		System.out.println(" peek() : "+s1.peek());		//fp
		System.out.println(" pop() : "+s1.pop());		//fp
		System.out.println(" empty() : "+s1.empty());		//true
	}
}
```

d) Hashtable class

This class implements a hashtable, which maps keys to values. Any non-null object can be used as a key or as a value.

i) public class Hashtable<K,V>

where, k is key

v is value

ii) public Hashtable()

- Constructs a new, empty hashtable with a default initial capacity (11)

iii) public Hashtable(int initialCapacity)

- Constructs a new, empty hashtable with the specified initial capacity

iv) public int size()

- Returns the number of keys in this hashtable.

v) public boolean isEmpty()

- Tests if this hashtable maps no keys to values.

vi) public Enumeration<K> keys()

- Returns an enumeration of the keys in this hashtable.

vii)public Enumeration<V> elements()

- Returns an enumeration of the values in this hashtable.

viii) public boolean contains(Object value)

- Tests if some key maps into the specified value in this hashtable.

ix) public V get(Object key)

- Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

x) public V put(K key,V value)

- Maps the specified key to the specified value in this hashtable. Neither the key nor the value can be null.

xi) public V remove(Object key)

- Removes the key (and its corresponding value) from this hashtable. This method does nothing if the key is not in the hashtable.

xii)public void clear()

- Clears this hashtable so that it contains no keys.

Example : abc.java

```
import java.util.*;
class abc
{
	public static void main(String s[])
	{
		Hashtable<Integer,String> h1 = new Hashtable<Integer,String>();
		h1.put(01,"fp");
		h1.put(05,"abc");
		h1.put(10,"xyz");
		System.out.println(" size() : "+h1.size());			//3
		Enumeration<Integer> e_int = h1.keys();
		while(e_int.hasMoreElements())
		{
			System.out.println(e_int.nextElement());
		}
		Enumeration<String> e_str = h1.elements();
		while(e_str.hasMoreElements())
		{
			System.out.println(e_str.nextElement());
		}
		System.out.println(" contains() : "+h1.contains("fp"));	//true
		System.out.println(" get() : "+h1.get(1));			//fp
		System.out.println(" remove() : "+h1.remove(10)); //xyz
		e_int = h1.keys();
		while(e_int.hasMoreElements())
		{
			System.out.println(e_int.nextElement());
		}
		System.out.println(" isEmpty() : "+h1.isEmpty());	//false
		h1.clear();
		System.out.println(" isEmpty() : "+h1.isEmpty());	//true
	}
}
```

e) StringTokenizer class

The StringTokenizer class allows an application to break a string into tokens.

StringTokenizer implements the enumeration interface. Therefore, given an input string, you can enumerate the individual tokens contained in it using StringTokenizer

The default set of delimiters consists of the white space char space, tab, newline, carriage return.

i) public StringTokenizer(String str)

- Constructs a string tokenizer for the specified string. The tokenizer uses the default delimiter set, which is " \t\n\r\f": the space character, the tab character, the newline character, the carriage-return character, and the form-feed character. Delimiter characters themselves will not be considered as tokens.
- **Parameters: str** - a string to be parsed.

ii) public StringTokenizer(String str,String delim)

- Constructs a string tokenizer for the specified string. The characters in the delim argument are the delimiters for separating tokens. Delimiter characters themselves will not be treated as tokens.

iii) public StringTokenizer(String str,String delim,boolean returnDelims)

- Constructs a string tokenizer for the specified string. All characters in the delim argument are the delimiters for separating tokens.
- If the returnDelims flag is true, then the delimiter characters are also returned as tokens. Each delimiter is returned as a string of length one. If the flag is false, the delimiter characters are skipped and only serve as separators between tokens.

iv) public boolean hasMoreElements()

- Returns the same value as the hasMoreTokens method. It exists so that this class can implement the Enumeration interface.

v) public Object nextElement()

- Returns the same value as the nextToken method, except that its declared return value is Object rather than String. It exists so that this class can implement the Enumeration interface.

vi) public boolean hasMoreTokens()

- Tests if there are more tokens available from this tokenizer's string. If this method returns true, then a subsequent call to nextToken with no argument will successfully return a token.
- **Returns:** true if and only if there is at least one token in the string after the current position; false otherwise.

vii)public String nextToken()

- Returns the next token from this string tokenizer.

- **Returns:** the next token from this string tokenizer.

Example : abc.java

```
import java.util.*;
class abcstok
{
	public static void main(String[] args)
	{
StringTokenizer st1 = new StringTokenizer("Hello, How are you fp ?");
		while(st1.hasMoreTokens())
		{
			System.out.println(st1.nextToken());
		}
		StringTokenizer st2 = new
StringTokenizer("Hello,+How+are+you+fp+?","+");
		while(st2.hasMoreTokens())
		{
			System.out.println(st2.nextToken());
		}
		StringTokenizer st3 = new
StringTokenizer("Hello,+How+are+you+fp+?","+",true);
		while(st3.hasMoreTokens())
		{
			System.out.println(st3.nextToken());
		}
	}
}
```

f) Date class

The class Date represents a specific instant in time, with millisecond precision.

i) public Date()

- creates an Date object with current date and time.

ii) public Date(long date)

- Convert long into the Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.

iii) public Object clone()

- Return a copy of this object.

iv) public long getTime()

- Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.

v) public boolean equals(Object obj)

- Compares two dates for equality. The result is true if and only if the argument is not null and is a Date object that represents the same point in time, to the millisecond, as this object.
- Thus, two Date objects are equal if and only if the getTime method returns the same long value for both.

vi) public int compareTo(Date anotherDate)

- Compares two Dates for ordering.
- **Returns:** the value 0 if the argument Date is equal to this Date; a value less than 0 if this Date is before the Date argument; and a value greater than 0 if this Date is after the Date argument.

Example : abc.java

```
class abc
{
	public static void main(String s[])
	{
		Date d = new Date();
		System.out.println(d);
	}
}
```

g) Calendar class

The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields .

Following are the fields of Calendar class.

Calendar.YEAR
Calendar.MONTH
Calendar.MINUTE
Calendar.AM_PM
Calendar.DAY_OF_MONTH
Calendar.DAY_OF WEEK
Calendar.DAY_OF_YEAR
Calendar.HOUR
Calendar.HOUR_OF_DAY

Calendar.MILLISECOND
Calendar.SECOND
Calendar.WEEK_OF_MONTH
Calendar.WEEK_OF_YEAR

i) public static Calendar getInstance()

- Gets a calendar using the default time zone and locale. The Calendar returned is based on the current time in the default time zone with the default locale.
- **Returns:** a Calendar object.

ii) public int get(int field)

- Returns the value of the given calendar field.

Example : abc.java

```
import java.util.*;
class abc
{
	public static void main(String s[])
	{
		Calendar c = Calendar.getInstance();
		System.out.println("Today's Year : "+c.get(Calendar.YEAR));
		System.out.println("Today's Month : "+c.get(Calendar.MONTH));
		System.out.println("Today's Date : "+c.get(Calendar.DATE));
		System.out.println(c.get(Calendar.HOUR));
		System.out.println(c.get(Calendar.MINUTE));
		System.out.println(c.get(Calendar.SECOND));
		System.out.println(c.get(Calendar.MILLISECOND));
	}
}
```

h) GregorianCalendar class

GregorianCalendar is a concrete subclass of Calendar and provides the standard calendar system used by most of the world.

public class GregorianCalendar extends Calendar

i) public GregorianCalendar()

- Constructs a default GregorianCalendar using the current time in the default time zone

ii) public GregorianCalendar(int year, int month, int dayOfMonth)

- Constructs a GregorianCalendar with the given date set in the default time zone with the default locale.
- **Parameters: year** - the value used to set the YEAR calendar field in the calendar.

  month - the value used to set the MONTH calendar field in the calendar. Month value is 0-based. e.g., 0 for January.

  dayOfMonth - the value used to set the DAY_OF_MONTH calendar field in the calendar.

iii) public GregorianCalendar(int year,int month,int dayOfMonth, int hourOfDay, int minute)

- Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.
- **Parameters: year** - the value used to set the YEAR calendar field in the calendar.

month - the value used to set the MONTH calendar field in the calendar. Month value is 0-based. e.g., 0 for January.

dayOfMonth - the value used to set the DAY_OF_MONTH calendar field in the calendar.

hourOfDay - the value used to set the HOUR_OF_DAY calendar field in the calendar.

minute - the value used to set the MINUTE calendar field in the calendar.

iv) Public GregorianCalendar(int year,int month,int dayOfMonth, int hourOfDay, int minute,int second)

- Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.
- **Parameters: year** - the value used to set the YEAR calendar field in the calendar.

  month - the value used to set the MONTH calendar field in the calendar. Month value is 0-based. e.g., 0 for January.

  dayOfMonth - the value used to set the DAY_OF_MONTH calendar field in the calendar.

  hourOfDay - the value used to set the HOUR_OF_DAY calendar field in the calendar.

  minute - the value used to set the MINUTE calendar field in the calendar.

  second - the value used to set the SECOND calendar field in the calendar.

v) public boolean isLeapYear(int year)

- Determines if the given year is a leap year. Returns true if the given year is a leap year.

- **Parameters: year** - the given year.
- **Returns:** true if the given year is a leap year; false otherwise.

i) Random class

An instance of this class is used to generate a stream of pseudorandom numbers

i) public Random()
- Creates a new random number generator.

ii) public int nextInt()
- Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence. The general contract of nextInt is that one int value is pseudorandomly generated and returned. All 2^{32} possible int values are produced with (approximately) equal probability.

iii) public int nextInt(int n)
- Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Example : abc.java

```
import java.util.*;
class abc
{
	public static void main(String s[])
	{
		Random rnd = new Random();
		for(int i=0;i<10;i++)
		{
			System.out.println(rnd.nextInt(10000));
		}
	}
}
```

3)java.io. package

This package provides support for Input/Output operation. Many programs can not achieve their goals without accessing external data.

A network connection, memory or physical file can be use through the java I/O classes.

A stream is use to handle these all.

A stream is a logically part of our system and it is stored information and can retrieve that stored information in it.

Java I/O system is linked Stream to the physical device.

a) File class

This class deals with directly with the file and the file system. This class does not specify hoe information is retrieved from or stored in files.

i) public File(String pathname)
- Creates a new File instance.

ii) public boolean createNewFile()throws IOException
- Create a new file if and only if a file with this name does not yet exist.

iii) public boolean delete()
- Deletes the file or directory. If delete the directory, then the directory must be empty in order to be deleted.

iv) public boolean exists()
- Tests whether the file or directory is exists.

v) public String getName()

Returns the name of the file or directory

vi) public String getPath()
- Returns the path of this file.

vii)public boolean isDirectory()
- Tests whether the invoking file is a directory.

viii) public boolean isFile()
- Tests whether the invoking file is a normal file.

ix) public long lastModified()
- Returns the time that the file was last modified.

x) public String[] list()

- Returns an array of strings naming the files and directories in the directory denoted by this pathname.

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws IOException
	{
		File f1 = new File("d:\\j2se\\programs\\file.txt");
		File f2 = new File("d:\\j2se\\programs");
		System.out.println(" createNewFile() : "+f1.createNewFile());
		System.out.println(" exists() : "+f1.exists());
		System.out.println(" getName() : "+f1.getName());
		System.out.println(" getPath() : "+f1.getPath());
		System.out.println(" isDirectory() : "+f1.isDirectory());
		System.out.println(" isDirectory() : "+f2.isDirectory());
		System.out.println(" isFile() : "+f1.isFile());
		System.out.println(" lastModified() : "+new
java.util.Date(f1.lastModified()));
		String nm[]=f2.list();
		for(int i=0;i<nm.length;i++)
		{
			System.out.println(nm[i]);
		}
		System.out.println(" delete() : "+f1.delete());
	}
}
```

b) The ByteStreams

It provides a wide environment for handling byte oriented I/O and can be used with any type of object including binary data.

There are following classes in ByteStreams.

InputStream
OutputStream
FileInputStream
FileOutputStream
FilterInputStream
BufferedInputStream

BufferedOutputStream
DataInputStream
DataOutputStream
ObjectInputStream
ObjectOutputStream
PrintStream

c) InputStream class

This abstract class is the superclass of all classes representing an input stream of bytes. Applications that need to define a subclass of InputStream must always provide a method that returns the next byte of input.

It means this abstract class is use to read bytes from a file.

public abstract class InputStream

i) public int available() throws IOException

- Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream.

ii) public void close() throws IOException

- Closes this input stream and releases any system resources associated with the stream.

iii) public abstract int read() throws IOException

- Reads the next byte of data from the input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.

d) OutputStream class

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to physical device.

Applications that need to define a subclass of OutputStream must always provide at least a method that writes one byte of output.

It means this abstract class is use to read bytes from a file.
public abstract class OutputStream

i) public void close()throws IOException

- Closes this output stream and releases any system resources associated with this stream. The general contract of close is that it closes the output stream. A closed stream cannot perform output operations and cannot be reopened.

ii) public abstract void write(int b)throws IOException

- Writes the specified byte to this output stream. The general contract for write is that one byte is written to the output stream. The byte to be written is the 8 low-order bits of the argument b. The 24 high-order bits of b are ignored.

iii) public void write(byte[] b)throws IOException

- Writes b.length bytes from the specified byte array to this output stream.

iv) public void flush()throws IOException

- Flushes this output stream and forces any buffered output bytes to be written out.

e) FileOutputStream class

This class is used to write bytes in file.

public class FileOutputStream extends OutputStream; It means this class can use methods of OutputStream abstract class to write bytes in file.

i) public FileOutputStream(String name)throws FileNotFoundException

- Creates an output file stream to write to the file with the specified name.

ii) publicFileOutputStream(String name,boolean append)throws FileNotFoundException

- Creates an output file stream to write to the file with the specified name. If the second argument is true, then bytes will be written to the end of the file rather than the beginning.

iii) public FileOutputStream(File file)throws FileNotFoundException

- Creates a file output stream to write to the file represented by the specified File object.

iv) publicFileOutputStream(File file,boolean append)throws FileNotFoundException

- Creates a file output stream to write to the file represented by the specified File object. If the second argument is true, then bytes will be written to the end of the file rather than the beginning.

Example - 1 : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		String s1 = " Hello, How are you ? fp ";
		File f = new File("write.txt");
		f.createNewFile();
		FileOutputStream fos1 = new FileOutputStream(f);
		fos1.write(s1.getBytes());
		fos1.close();
		FileOutputStream fos2 = new FileOutputStream(f);
		String s2 = " ya i am fine !!!!!!! ";
		fos2.write(s2.getBytes());
		fos2.close();
	}
}
```

In above program only **" ya I am fine !!!!!!! "** is written in file. Because when file is again open for writing then the writing start at beginning. It means the file is not open in append mode.

Example – 2:abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		String s1 = " Hello, How are you ? fp ";
		File f = new File("write.txt");
		f.createNewFile();
		FileOutputStream fos1 = new FileOutputStream(f);
		fos1.write(s1.getBytes());
		fos1.close();
		FileOutputStream fos2 = new FileOutputStream(f,true);
```

```
            String s2 = " ya i am fine !!!!!!! ";
            fos2.write(s2.getBytes());
            fos2.close();
        }
}
```

In above program only **" Hello, How are you ? fp ya I am fine !!!!!!! "** is written in file. Because when file is again open for writing then the writing start at end. It means the file is open in append mode.

f) FileInputStream class

This class is used to read bytes from the file.

public class FileInputStream extends InputStream; It means this class can use methods of InputStream abstract class to read bytes from file.

i) public FileInputStream(String name)throws FileNotFoundException

- Creates a FileInputStream by opening a connection to an actual file, the file named by the path name name in the file system.

ii) public FileInputStream(File file)throws FileNotFoundException

- Creates a FileInputStream by opening a connection to an actual file, the file named by the File object file in the file system.

Example : abc.java

```
import java.io.*;
class abc
{
        public static void main(String s[]) throws Exception
        {
                String s = " Hello, How are you ? fp ";
                File f = new File("write.txt");
                f.createNewFile();
                FileOutputStream fos = new FileOutputStream(f);
                fos.write(s.getBytes());
                fos.close();
                FileInputStream fis = new FileInputStream(f);
                int can=fis.available();
                for(int i=0;i<can;i++)
                {
                        System.out.print((char)fis.read());
```

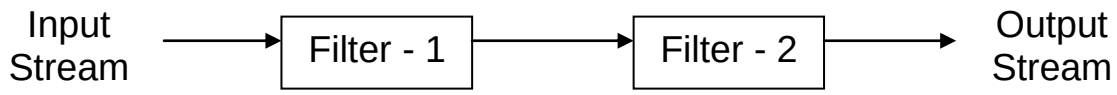
```
            }
            fis.close();
        }
}
```

g) FilteredByteStream classes

Filter streams are simply wrappers around underlying I/O streams that transparently provide extended level of functionality.
This streams are typically accessed by methods that are expecting generic stream which is sub-classes of the Filtered streams.

In FilteredByteStream classes following 2 classes are there:

1) FilterInputStream
2) FilterOutputStream



The FilterInputStream and FilterOutputStream abstract classes that provides basic capability to create InputStream and OutputStream for filtering I/O in a number of ways.

This streams are known as filters because they shift between InputStream to OutputStream and perform some option processing on the data when they are transfer.

h) BufferedOutputStream class

The class implements a buffered output stream. By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written.

i) public BufferedOutputStream(OutputStream out)

- Creates a new buffered output stream to write data to the specified underlying output stream.
- **Parameters: out** - the underlying output stream.

ii) public BufferedOutputStream(OutputStream out,int size)

- Creates a new buffered output stream to write data to the specified underlying output stream with the specified buffer size.

- **Parameters: out** - the underlying output stream.
  size - the buffer size.

iii) public void write(int b) throws IOException

- Writes the specified byte to this buffered output stream.

iv) public void write(byte[] b,int off,int len) throws IOException

- Writes len bytes from the specified byte array starting at offset off to this buffered output stream.

v) public void flush() throws IOException

- Flushes this buffered output stream. This forces any buffered output bytes to be written out to the underlying output stream.

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("test.txt");
		f.createNewFile();
		FileOutputStream fos = new FileOutputStream(f);
		BufferedOutputStream bos = new BufferedOutputStream(fos);
		bos.write("Line one".getBytes());
		bos.write(65);
		bos.close();
	}
}
```

i) BufferedInputStrean class

A BufferedInputStream adds functionality to another input stream-namely, the ability to buffer the input and to support the mark and reset methods. When the BufferedInputStream is created, an internal buffer array is created. As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time.

i) public BufferedInputStream(InputStream in)

- Creates a BufferedInputStream and saves its argument, the input stream in, for later use. An internal buffer array is created and stored in buf.
- **Parameters: in** - the underlying input stream.

ii) public BufferedInputStream(InputStream in,int size)

- Creates a BufferedInputStream with the specified buffer size, and saves its argument, the input stream in, for later use. An internal buffer array of length size is created and stored in buf.
- **Parameters: in** - the underlying input stream.
  size - the buffer size.

iii) public int available() throws IOException

- Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream.

iv) public void close() throws IOException

- Closes this input stream and releases any system resources associated with the stream.

v) public int read() throws IOException

- Reads the next byte of data from the input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.

Example : abc.java

```
import java.io.*;
class abcbis
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("test.txt");
		f.createNewFile();
		FileOutputStream fos = new FileOutputStream(f);
		BufferedOutputStream bos = new BufferedOutputStream(fos);
		bos.write(" Hello, How are you ? fp ".getBytes());
		bos.close();
		FileInputStream fis = new FileInputStream(f);
		BufferedInputStream bis = new BufferedInputStream(fis);
		while(bis.available() > 0)
		{
			System.out.print((char)bis.read());
		}
		bis.close();
	}
}
```

Program to read content of one file and write in another file.

Example : abc.java

```
import java.io.*;
class abcio
{
	public static void main(String s[]) throws Exception
	{
		File fin = new File("in.txt");
		File fout = new File("out.txt");
		fin.createNewFile();
		fout.createNewFile();
		FileOutputStream fos = new FileOutputStream(fin);
		fos.write( " Hello, How are you ? fp ".getBytes());
		fos.close();
		BufferedInputStream bisin = new BufferedInputStream(new
FileInputStream(fin));
		BufferedOutputStream bosout = new BufferedOutputStream(new
FileOutputStream(fout));
		int i;
		do
		{
			i = bisin.read();
			if (i != -1)
			bosout.write(i);
		} while (i != -1);
		bisin.close();
		bosout.close();
	}
}
```

Program to get input from user in console.

Example : abc.java

```
import java.io.*;
class abc
{
	public static String readLine()
	{
		StringBuffer res = new StringBuffer();
		try
		{
		   BufferedInputStream bis = new BufferedInputStream(System.in);
			int in = 0;
```

```
                char inChar;
                do
                {
                        in = bis.read();
                        inChar = (char)in;
                        if ((in != -1) & (in != '\n') & (in != '\r'))
                        {
                                res.append(inChar);
                        }
                }while ((in != -1) & (inChar != '\n') & (in != '\r'));
                bis.close();
                return res.toString();
        }
        catch (IOException e)
        {
                System.out.println("Exception: " + e);
                return null;
        }
    }
    public static void main(String s[])
    {
        System.out.print("\nWhat is your name ? :  ");
        String input = readLine();
        System.out.println("\nHello, " + input);
    }
}
```

j) DataOutputStream class

A DataOutputStream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in.

i) public DataOutputStream(OutputStream out)

- Creates a new data output stream to write data to the specified underlying output stream. The counter written is set to zero.
- **Parameters: out** - the underlying output stream, to be saved for later use.

ii) public void write(int b)throws IOException

- Writes the specified byte (the low eight bits of the argument b) to the underlying output stream.

iii) public final void writeBoolean(boolean v)throws IOException

- Writes a boolean to the underlying output stream as a 1-byte value. The value true is written out as the value (byte)1; the value false is written out as the value (byte)0.

iv) public final void writeChar(int v)throws IOException

- Writes a char to the underlying output stream as a 2-byte value, high byte first.

v) public final void writeInt(int v)throws IOException

- Writes an int to the underlying output stream as four bytes, high byte first.

vi) public final void writeUTF(String str)throws IOException

- Writes a string to the underlying output stream.

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("demo.txt");
		f.createNewFile();
		FileOutputStream fos = new FileOutputStream(f);
		DataOutputStream dos = new DataOutputStream(fos);
		dos.writeBytes("Hello fp !!! How are you ? ");
		dos.close();
	}
}
```

k) DataInputStream class

A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way. An application uses a data output stream to write data that can later be read by a data input stream.

i) public DataInputStream(InputStream in)

- Creates a DataInputStream that uses the specified underlying InputStream.
- **Parameters: in** - the specified input stream

ii) boolean readBoolean() throws IOException

- Reads one input byte and returns true if that byte is nonzero, false if that byte is zero. This method is suitable for reading the byte written by the writeBoolean method of interface DataOutput.
- **Returns:** the boolean value read.

iii) char readChar() throws IOException

- Reads two input bytes and returns a char value

iv) int readInt() throws IOException

- Reads four input bytes and returns an int value.

v) String readUTF() throws IOException

- Reads a string

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("demo.txt");
		f.createNewFile();
		FileOutputStream fos = new FileOutputStream(f);
		DataOutputStream dos = new DataOutputStream(fos);
		dos.writeBoolean(true);
		dos.writeInt(10);
		dos.writeUTF(" Hello fp !!!! How are you ? ");
		dos.close();
		FileInputStream fis = new FileInputStream(f);
		DataInputStream dis = new DataInputStream(fis);
		System.out.println(dis.readBoolean());
		System.out.println(dis.readInt());
		System.out.println(dis.readUTF());
	}
}
```

l) ObjectOutputStream class

An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream. The objects can be read (reconstituted) using an ObjectInputStream. Persistent storage of objects can be accomplished by using a file for the stream. If the

stream is a network socket stream, the objects can be reconstituted on another host or in another process.

Only objects that support the java.io.Serializable interface can be written to streams. The class of each serializable object is encoded including the class name and signature of the class, the values of the object's fields and arrays, and the closure of any other objects referenced from the initial objects.

i) public ObjectOutputStream(OutputStream out) throws IOException
- Creates an ObjectOutputStream that writes to the specified OutputStream.

ii) public final void writeObject(Object obj) throws IOException
- Write the specified object to the ObjectOutputStream.

iii) public void write(int val) throws IOException
- Writes a byte.

iv) public void write(byte[] buf) throws IOException
- Writes an array of bytes.

v) public void flush() throws IOException
- Flushes the stream. This will write any buffered output bytes and flush through to the underlying stream.

vi) public void close() throws IOException
- Closes the stream. This method must be called to release any resources associated with the stream.

vii)public void writeBoolean(boolean val) throws IOException
- Writes a boolean.

viii) public void writeChar(int val) throws IOException
- Writes a 16 bit char.

ix) public void writeInt(int val) throws IOException
- Writes a 32 bit int.

x) public void writeChars(String str) throws IOException
- Writes a String as a sequence of chars.

xi) public void writeUTF(String str) throws IOException
- Primitive data write of this String

Example : abc.java

```
import java.io.*;
class Person implements Serializable
{
	String firstName;
	String lastName;
	int age;
	Person() { }
	public String getFirstName()
	{
		return firstName;
	}
	public void setFirstName(String firstName)
	{
		this.firstName = firstName;
	}
	public String getLastName()
	{
		return lastName;
	}
	public void setLastName(String lastName)
	{
		this.lastName = lastName;
	}
	public int getAge()
	{
		return age;
	}
	public void setAge(int age)
	{
		this.age = age;
	}
	public String toString()
	{
		StringBuffer buffer = new StringBuffer();
		buffer.append(firstName);
		buffer.append("\n");
		buffer.append(lastName);
		buffer.append("\n");
		buffer.append(age);
		buffer.append("\n");
		return buffer.toString();
```

```
	}
}
class abc
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("demo.txt");
		f.createNewFile();
		FileOutputStream fos = new FileOutputStream(f);
		ObjectOutputStream os = new ObjectOutputStream(fos);
		Person person = new Person();
		person.setFirstName("A");
		person.setLastName("B");
		person.setAge(38);
		os.writeObject(person);
		person = new Person();
		person.setFirstName("C");
		person.setLastName("D");
		person.setAge(22);
		os.writeObject(person);
	}
}
```

m) ObjectInputStream class

An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.

ObjectOutputStream and ObjectInputStream can provide an application with persistent storage for graphs of objects when used with a FileOutputStream and FileInputStream respectively. ObjectInputStream is used to recover those objects previously serialized.

ObjectInputStream ensures that the types of all objects in the graph created from the stream match the classes present in the Java Virtual Machine. Classes are loaded as required using the standard mechanisms.

Only objects that support the java.io.Serializable or java.io.Externalizable interface can be read from streams.

i) **public ObjectInputStream(InputStream in) throws IOException**

- Creates an ObjectInputStream that reads from the specified InputStream. A serialization stream header is read from the stream and verified.

ii) **public final Object readObject() throws IOException, ClassNotFoundException**

- Read an object from the ObjectInputStream. The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all of its supertypes are read.

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		FileInputStream fis = new FileInputStream("demo.txt");
		ObjectInputStream is = new ObjectInputStream(fis);
		Object obj = null;
		while ((obj = is.readObject()) != null)
		{
			if (obj instanceof Person)
			{
				System.out.println(((Person) obj).toString());
			}
		}
		is.close();
	}
}
class Person implements Serializable
{
	String firstName;
	String lastName;
	int age;
	Person() {}
	public String getFirstName()
	{
		return firstName;
	}
	public void setFirstName(String firstName)
	{
		this.firstName = firstName;
```

```
	}
	public String getLastName()
	{
		return lastName;
	}
	public void setLastName(String lastName)
	{
		this.lastName = lastName;
	}
	public int getAge()
	{
		return age;
	}
	public void setAge(int age)
	{
		this.age = age;
	}
	public String toString()
	{
		StringBuffer buffer = new StringBuffer();
		buffer.append(firstName);
		buffer.append("\n");
		buffer.append(lastName);
		buffer.append("\n");
		buffer.append(age);
		buffer.append("\n");
		return buffer.toString();
	}
}
```

n) PrintStream class

A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently. Two other features are provided as well. Unlike other output streams, a PrintStream never throws an IOException; instead, exceptional situations merely set an internal flag that can be tested via the checkError method. Optionally, a PrintStream can be created so as to flush automatically; this means that the flush method is automatically invoked after a byte array is written, one of the println methods is invoked, or a newline character or byte ('\n') is written.

i) public PrintStream(File file)throws FileNotFoundException

- Creates a new print stream, without automatic line flushing, with the specified file.

ii) public PrintStream(OutputStream out)

- Creates a new print stream. This stream will not flush automatically.
- **Parameters: out** - The output stream to which values and objects will be printed

iii) public PrintStream(OutputStream out,boolean autoFlush)

- Creates a new print stream.
- **Parameters: out** - The output stream to which values and objects will be printed

  autoFlush - A boolean; if true, the output buffer will be flushed whenever a byte array is written, one of the println methods is invoked, or a newline character or byte ('\n') is written

iv) public PrintStream(String fileName)throws FileNotFoundException

- Creates a new print stream, without automatic line flushing, with the specified file name.
- **Parameters: fileName** - The name of the file to use as the destination of this print stream. If the file exists, then it will be truncated to zero size; otherwise, a new file will be created. The output will be written to the file and is buffered.

v) public void flush()

- Flushes the stream. This is done by writing any buffered output bytes to the underlying output stream and then flushing that stream.

vi) public void close()

- Closes the stream. This is done by flushing the stream and then closing the underlying output stream.

vii)public void write(int b)

- Writes the specified byte to this stream.

viii) public void print(char c)

- Prints a character.

ix) public void print(int i)

- Prints an integer.

x) public void print(char[] s)

- Prints an array of characters.

xi) public void print(String s)

- Prints a string.

xii)public void print(Object obj)

- Prints an object.

xiii) Public void println(char x)

- Prints a character and then terminate the line.

xiv) public void println(String x)

- Prints a String and then terminate the line.

xv)public void println(Object x)

- Prints an Object and then terminate the line.

o) CharacterStream classes

While the byte streams classes provide sufficient functionality to handle any type of I/O operation, they can not work directly with Unicode characters.

One of the main purpose of java is to support the " write once run anywhere " philosophy. It was necessary to include direct I/O support for characters.

There are following classes in CharacterStreams.

Reader
Writer
FileReader
FileWriter
BufferedReader
BufferedWriter
PrintWriter

p) Writer class

Abstract class for writing to character streams.

i) public Writer append(char c) throws IOException

- Appends the specified character to this writer.

ii) public void write(String str) throws IOException

- Writes a string.

iii) public Writer append(CharSequence csq) throws IOException

- Appends the specified character sequence to this writer.

iv) public Writer append(char c) throws IOException

- Appends the specified character to this writer.

v) public abstract void flush() throws IOException

- Flushes the stream.

vi) public abstract void close() throws IOException

- Closes the stream, flushing it first.

q) Reader class

Abstract class for reading character streams.

i) public int read() throws IOException

- Reads a single character.

ii) public long skip(long n) throws IOException

- Skips characters.

iii) public boolean ready() throws IOException

- Tells whether this stream is ready to be read.

iv) public abstract void close() throws IOException

- Closes the stream and releases any system resources associated with it.

r) FileWriter class

Abstract class for writing filtered character streams.

public abstract class FilterWriter extends Writer

i) protected FilterWriter(Writer out)

- Create a new filtered writer.
- **Parameters: out** - a Writer object to provide the underlying stream.

ii) public void write(int c) throws IOException

- Writes a single character.

iii) public void flush() throws IOException

- Flushes the stream.

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("demo.txt");
		f.createNewFile();
		FileWriter fw = new FileWriter("demo.txt");
		char c = 'A';
		fw.write(c);
		fw.write(" Hello fp, How are you ? ");
		fw.close();
	}
}
```

s) FileReader class

Abstract class for reading filtered character streams.

public abstract class FilterReader extends Reader

i) protected FilterReader(Reader in)

- Creates a new filtered reader.
- **Parameters: in** - a Reader object providing the underlying stream.

ii) public long skip(long n) throws IOException

- Skips characters.

iii) public boolean ready() throws IOException

- Tells whether this stream is ready to be read.

Example : abc.java

```
import java.io.*;
class abc
{
```

```
	public static void main(String s[]) throws Exception
	{
		FileReader fr = new FileReader("demo.txt");
		int ch;
		do
		{
			ch = fr.read();
			if (ch != -1)
			System.out.println((char) ch);
		}while (ch != -1);
		fr.close();
	}
}
```

t) BufferedWriter class

Writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.

The buffer size may be specified, or the default size may be accepted. The default is large enough for most purposes.

i) public BufferedWriter(Writer out)

- Creates a buffered character-output stream that uses a default-sized output buffer.
- **Parameters:** out - A Writer

ii) public void write(int c)throws IOException

- Writes a single character.

iii) public void newLine()throws IOException

- Writes a line separator.

iv) public void flush()throws IOException

- Flushes the stream.

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
```

```
            File f = new File("demo.txt");
            f.createNewFile();
            FileWriter fw = new FileWriter("demo.txt");
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write(" Hello fp ");
            bw.newLine();
            bw.write(" How are you ? ");
            bw.close();
        }
}
```

u) BufferedReader class

Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

The buffer size may be specified, or the default size may be used. The default is large enough for most purposes.

i) public BufferedReader(Reader in)

- Creates a buffering character-input stream that uses a default-sized input buffer.
- **Parameters: in** - A Reader.

ii) public int read()throws IOException

- Reads a single character.

iii) public String readLine()throws IOException

- Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.
- **Returns:** A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Example : abc.java

```
import java.io.*;
class abc
{
        public static void main(String s[]) throws Exception
        {
                FileReader fr = new FileReader (new File("demo.txt"));
                BufferedReader br = new BufferedReader (fr);
```

```
            String line = br.readLine();
            while (line != null)
            {
                  System.out.println(line);
                  line = br.readLine();
            }
            br.close();
      }
}
```

v) PrintWriter class

Prints formatted representations of objects to a text-output stream.

i) public PrintWriter(Writer out)
- Creates a new PrintWriter, without automatic line flushing.
- **Parameters: out** - A character-output stream

ii) public PrintWriter(OutputStream out)
- Creates a new PrintWriter, without automatic line flushing, from an existing OutputStream.

iii) public PrintWriter(String fileName) throws FileNotFoundException
- Creates a new PrintWriter, without automatic line flushing, with the specified file name.

iv) public PrintWriter(File file) throws FileNotFoundException
- Creates a new PrintWriter, without automatic line flushing, with the specified file.

v) public void flush()
- Flushes the stream.

vi) public void close()
- Closes the stream and releases any system resources associated with it. Closing a previously closed stream has no effect.

vii)public void write(int c)
- Writes a single character.

viii) public void write(String s)
- Writes a string.

ix) public void println(String x)

- Prints a String and then terminates the line.

x) public void println(Object x)

- Prints an Object and then terminates the line.

Example : abc.java

```
import java.io.*;
class abc
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("test.txt");
		f.createNewFile();
		PrintWriter pw = new PrintWriter(f);
		pw.println("Hello fp, How are you ? ");
		pw.println(1234);
		pw.close();
	}
}
```

w) RandomAccessFile class

Instances of this class support both reading and writing to a random access file.

A random access file behaves like a large array of bytes stored in the file system. There is a kind of cursor, or index into the implied array, called the *file pointer*; input operations read bytes starting at the file pointer and advance the file pointer past the bytes read.

If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. Output operations that write past the current end of the implied array cause the array to be extended.

Threre are 4 type of mode in RandomAccessFile.

1)"r" Open for reading only. Invoking any of the write methods of the resulting object will cause an IOException to be thrown.

2)"rw" Open for reading and writing. If the file does not already exist then an attempt will be made to create it.

3)"rws" Open for reading and writing, as with "rw", and also require that every

update to the file's content or metadata be written synchronously to the underlying storage device.

4)"rwd" Open for reading and writing, as with "rw", and also require that every update to the file's content be written synchronously to the underlying storage device.

i) public RandomAccessFile(File file,String mode) throws FileNotFoundException

- Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.

ii) publicRandomAccessFile(String name,String mode)throwsFileNotFound Exception

- Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

iii) public int read() throws IOException

- Reads a byte of data from this file.

iv) public final void readFully(byte[] b) throws IOException

- Reads b.length bytes from this file into the byte array, starting at the current file pointer. This method reads repeatedly from the file until the requested number of bytes are read.

v) public void write(int b) throws IOException

- Writes the specified byte to this file. The write starts at the current file pointer.

vi) public void write(byte[] b) throws IOException

- Writes b.length bytes from the specified byte array to this file, starting at the current file pointer.

vii)public long getFilePointer() throws IOException

- Returns the current offset in this file.

viii) public void seek(long pos) throws IOException

- Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. The offset may be set beyond the end of the file. Setting the offset beyond the end of the file does not change the file length. The file length will change only by writing after the offset has been set beyond the end of the file.

ix) public long length() throws IOException

- Returns the length of this file.

x) public void close() throws IOException

- Closes this random access file stream and releases any system resources associated with the stream. A closed random access file cannot perform input or output operations and cannot be reopened.
- If this file has an associated channel then the channel is closed as well.

xi) public final boolean readBoolean() throws IOException

- Reads a boolean from this file.

xii)public final char readChar() throws IOException

- Reads a character from this file.

xiii) public final int readInt()throws IOException

- Reads a signed 32-bit integer from this file.

xiv) public final String readLine() throws IOException

- Reads the next line of text from this file.

xv)public final void writeBoolean(boolean v) throws IOException

- Writes a boolean to the file as a one-byte value. The value true is written out as the value (byte)1; the value false is written out as the value (byte)0. The write starts at the current position of the file pointer.

xvi) public final void writeChar(int v) throws IOException

- Writes a char to the file as a two-byte value, high byte first. The write starts at the current position of the file pointer.

xvii) public final void writeInt(int v) throws IOException

- Writes an int to the file as four bytes, high byte first. The write starts at the current position of the file pointer.

xviii) public final void writeChars(String s) throws IOException

- Writes a string to the file as a sequence of characters. Each character is written to the data output stream as if by the writeChar method. The write starts at the current position of the file pointer.

Example – 1:abc.java

```
import java.io.*;
class abc
```

```
{
	public static void main(String s[]) throws Exception
	{
		File f = new File("demo.txt");
		f.createNewFile();
		RandomAccessFile raf = new RandomAccessFile(f, "rw");
		raf.writeInt(10);
		raf.writeInt(43);
		raf.writeInt(88);
		raf.writeInt(455);
		// change the 3rd integer from 88 to 99
		raf.seek((3 - 1) * 4);
		raf.writeInt(99);
		raf.seek(0); // go to the first integer
		int i = raf.readInt();
		while (i != -1)
		{
			System.out.println(i);
			i = raf.readInt();
		}
		raf.close();
	}
}
```

x) StreamTokenizer class

The StreamTokenizer class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states. The stream tokenizer can recognize identifiers, numbers, quoted strings, and various comment styles.

public int type

After a call to the nextToken method, this field contains the type of the token just read. For a single character token, its value is the single character, converted to an integer. For a quoted string token, its value is the quote character. Otherwise, its value is one of the following:

TT_WORD indicates that the token is a word.

TT_NUMBER indicates that the token is a number.

TT_EOL indicates that the end of line has been read. The field can only have this value if the eolIsSignificant method has been called with the argument true.

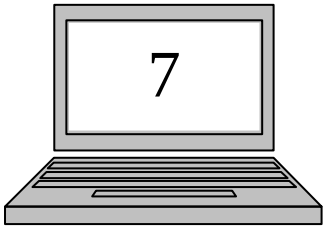
TT_EOF indicates that the end of the input stream has been reached.

i) public StreamTokenizer(Reader r)

- Create a tokenizer that parses the given character stream.
- **Parameters: r** - a Reader object providing the input stream.

Example : abc.java

```
import java.io.*;
class abcst
{
	public static void main(String[] args) throws Exception
	{
		File f = new File("demo.txt");
		f.createNewFile();
		FileWriter fw = new FileWriter(f);
		fw.write(" Hello, fp !!! How are you ? 123 321");
		fw.close();
		int wordCount = 0, numberCount = 0;
		FileReader fr = new FileReader(f);
		StreamTokenizer sTokenizer = new StreamTokenizer(fr);
		while (sTokenizer.nextToken() != StreamTokenizer.TT_EOF)
		{
			if (sTokenizer.ttype == StreamTokenizer.TT_WORD)
			{
				wordCount++;
			}
		else if (sTokenizer.ttype == StreamTokenizer.TT_NUMBER)
			{
		      numberCount++;
			}
		}
	   System.out.println("Number of words in file: " + wordCount);
	System.out.println("Number of numbers in file: " + numberCount);
	}
}
```



CHAPTER SEVEN

Applet

IN THIS CHAPTER

- Applet
- Applet Class
- Applet Lifecycle
- Appletcontext Interface
- An <Applet> Tag
- Passing parameter to Applet
- Graphic Class(java.awt.graphics)

1) Applet

Applets are small applications that are accessed on a web page, transported over the Internet, automatically installed, and run as part of a Web page.

In java every applet must be a subclass of java.applet.Applet.

An applet is not console base programs, it is a GUI (Graphical User Interface) programs. Applets work with user interactions/event.

Let's take an example for detailed understanding of applets.

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=200 height=60>
        </applet>
*/
public class abc extends Applet
{
        ……;
        ……;
        ……;
}
```

Save as the above program: abc.java

If you want to create an applet application you must import 2 packages. First is awt (AWT – Abstract Window Toolkit) to create GUI. The awt package provides classes (like Button, Label, etc) to create GUI with applet. Second is applet package which contains the Applet class.

Next you write an <applet> tag must be in comment (only in java file) which is using to run our applet in java enabled web browser. The <applet> tag has 3 attributes

code	:	applet class name
width	:	width of display area used by applet
height	:	height of display area used by applet

The next line in the program declares the class abc. This class must be declared as public, because it will be accessed by code that is outside the program.

Now compile the above program same as console base program

javac abc.java

The applet program runs in two ways:

using appletviewer

in web browser

To run the above applet program simply write:

appletviewer abc.java

And to run above program in web browser follow the below steps

- Save as the above program in html format. (abc.html)
- Remove the imports, other java codes and only comments of an <applet> tag.
- Put the <applet> tag in <body> tag of abc.html file
- And double click on abc.html file
- This html file and class file must be in same directory.

abc.html

```
<html>
    <body>
        <applet code="abc" width=200 height=60>
        </applet>
    </body>
</html>
```

2) Applet class

java.lang.Object > java.awt.Component > java.awt.Container > java.awt.Panel > java.applet.Applet

Applet provides all of the necessary support for window-based activities.

a) public Applet() throws HeadlessException

- Constructs a new Applet.
- Many methods in java.applet.Applet may be invoked by the applet only after the applet is fully constructed; applet should avoid calling methods in java.applet.Applet in the constructor.

b) public void init()

- Called by the browser or applet viewer to inform this applet that it has been loaded into the system. It is always called before the first time that the start method is called.
- A subclass of Applet should override this method if it has initialization to perform. For example, an applet with threads would use the init method to

create the threads and the destroy method to kill them. The implementation of this method provided by the Applet class does nothing.

c) public void start()

- Called by the browser or applet viewer to inform this applet that it should start its execution. It is called after the init method and each time the applet is revisited in a Web page.
- A subclass of Applet should override this method if it has any operation that it wants to perform each time the Web page containing it is visited. For example, an applet with animation might want to use the start method to resume animation, and the stop method to suspend the animation.

d) public void stop()

- Called by the browser or applet viewer to inform this applet that it should stop its execution. It is called when the Web page that contains this applet has been replaced by another page, and also just before the applet is to be destroyed.
- A subclass of Applet should override this method if it has any operation that it wants to perform each time the Web page containing it is no longer visible. For example, an applet with animation might want to use the start method to resume animation, and the stop method to suspend the animation.

e) public void destroy()

- Called by the browser or applet viewer to inform this applet that it is being reclaimed and that it should destroy any resources that it has allocated. The stop method will always be called before destroy.
- A subclass of Applet should override this method if it has any operation that it wants to perform before it is destroyed. For example, an applet with threads would use the init method to create the threads and the destroy method to kill them.

f) public URL getCodeBase()

- Gets the base URL. This is the URL of the directory which contains this applet.
- **Returns:** the base URL of the directory which contains this applet.

g) public URL getDocumentBase()

- Gets the URL of the document in which this applet is embedded. For example, suppose an applet is contained within the document:
- **Returns:** the URL of the document that contains this applet.

h) public void repaint(int x,int y,int width,int height)

- Repaints the specified rectangle of this component.

i) public void print(Graphics g)

- Prints this component. Applications should override this method for components that must do special processing before being printed or should be printed differently than they are painted.

j) public void setBackground(Color c)
- Sets the background color of this component.

k) public void setForeground(Color c)
- Sets the foreground color of this component.

l) public Color getBackground()
- Gets the background color of this component.
- **Returns:** this component's background color; if this component does not have a background color, the background color of its parent is returned

m) public Color getForeground()
- Gets the foreground color of this component.
- **Returns:** this component's foreground color; if this component does not have a foreground color, the foreground color of its parent is returned

n) public void showStatus(String msg)
- Requests that the argument string be displayed in the "status window". Many browsers and applet viewers provide such a window, where the application can inform users of its current state.
- **Parameters: msg** - a string to display in the status window.

o) public String getParameter(String name)
- Returns the value of the named parameter in the HTML tag.
- **Parameters: name** - a parameter name. (case insensitive)
- **Returns:** the value of the named parameter, or null if not set.

3) Applet lifecycle

An Applet lifecycle means execution of Applet in browser or applet viewer. In Applet lifecycle there are 4 methods participate.

init()
start()
stop()
destroy()

Above 4 methods are belong to Applet class.

paint()

And above paint() method belongs to AWT Component class. This method is not a part of Applet lifecycle.

Consider the following example of Applet lifecycle.

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=300 height=100>
        </applet>
*/
public class abc extends Applet
{
        public void init()
        {
                // Called first.
        }
        public void start()
        {
                // Called second, after init(). Also called whenever the applet is
                restarted.
        }
        public void stop()
        {
                // Called when the applet is stopped.
        }
        public void destroy()
        {
                // Called when applet is terminated.  This is the last method executed.
        }
        public void paint(Graphics g)
        {
                // Called when an applet's window must be restored.
        }
}
```

In Applet lifecycle the init(), start() and paint() method called when Applet begins and stop() and destroy() methods call when Applet is over or terminate.

init()

This is the first method to be called during Applet execution. This method is use to initialize the variables and objects. This method is called at only one time during Applet lifecycle.

start()
This method is called just after the init() method. And it also call when the applet is restarted. This method calls every time when applet displayed on screen.

paint()
This method is used to print the graphical content on our applet like displaying output, drawing shapes or many more graphical content which we will be study in AWT Graphics class. This method also calls every time when applet displayed on screen.

stop()
This method calls when applet lost the focus. When stop() method calls the applet may be running. This method calls before the destroy() method will call.

destroy()
This method calls when applet completely removes from memory.

Example – 1: abc.java

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=500 height=500>
        </applet>
*/
public class abc extends Applet
{
        StringBuffer sb = new StringBuffer();
        public void init()
        {
                sb.append(" first init() called ");
        }
        public void start()
        {
                sb.append(" Second start() called ");
        }
        public void stop()
        {
                sb.append(" Third stop() called ");
        }
```

```
	public void paint(Graphics g)
	{
		g.drawString(sb.toString(),10,10);
	}
}
```

Example – 2 :abc.java

```
import java.awt.*;
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
	g.drawString(" getCodeBase() : "+getCodeBase().toString(),10,10);
	g.drawString(" getDocumentBase() :   "+
	getDocumentBase().toString(),10,20);
	}
}
```

Exampe – 3: abc.java

```
import java.awt.*;
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void init()
	{
		setBackground(Color.red);
		setForeground(Color.yellow);
	}
	public void paint(Graphics g)
	{
		g.drawString(" Hello fp !!!!!! How are you ",10,10);
		showStatus(" Ya i am fine ");
	}
}
```

4) AppletContext interface

This interface corresponds to an applet's environment: the document containing the applet and the other applets in the same document.
The methods in this interface can be used by an applet to obtain information about its environment.

a) **AudioClip getAudioClip(URL url)**
- Creates an audio clip.
- **Parameters: url** - an absolute URL giving the location of the audio clip.
- **Returns:** the audio clip at the specified URL.

b) **Image getImage(URL url)**
- Returns an Image object that can then be painted on the screen. The url argument that is passed as an argument must specify an absolute URL.

c) **Applet getApplet(String name)**
- Finds and returns the applet in the document represented by this applet context with the given name. The name can be set in the HTML tag by setting the name attribute.

d) **Enumeration<Applet> getApplets()**
- Finds all the applets in the document represented by this applet context.
- **Returns:** an enumeration of all applets in the document represented by this applet context.

e) **void showDocument(URL url)**
- Requests that the browser or applet viewer show the Web page indicated by the url argument. The browser or applet viewer determines which window or frame to display the Web page. This method may be ignored by applet contexts that are not browsers.
- **Parameters:** url - an absolute URL giving the location of the document.

f) **void showStatus(String status)**
- Requests that the argument string be displayed in the "status window". Many browsers and applet viewers provide such a window, where the application can inform users of its current state.
- **Parameters: status** - a string to display in the status window.

5) An <Applet> tag

< **APPLET**
[**CODEBASE** = codebaseURL]
CODE = appletFile
[**ALT** = alternateText]
[**NAME** = appletInstanceName]
WIDTH = pixels **HEIGHT** = pixels
[**ALIGN** = alignment]
[**VSPACE** = pixels] [**HSPACE** = pixels]
>
[< **PARAM NAME** = AttributeName **VALUE** = AttributeValue>]
[< **PARAM NAME** = AttributeName2 **VALUE** = AttributeValue>]
</**APPLET**>

CODEBASE
It is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag).

CODE
Name of the applet compiled class file.

ALT
It is an optional attribute used to specify a short text message that should be displayed if the browser understands the APPLET tag but can't currently run Java applets. This is distinct from the alternate HTML you provide for browsers that don't support applets.

NAME
NAME is an optional attribute used to specify a name for the applet instanc. Applets must be named in order for other applets on the same page to find them by name and communicate with them. To obtain an applet by name, use getApplet(),which is defined by the AppletContext interface.

WIDTH
Width of applet displayed area on screen.

HEIGHT
Height of applet displayed area on screen.

ALIGN
ALIGN is an optional attribute that specifies the alignment of the applet. This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE and ABSBOTTOM.

VSPACE & HSPACE
These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet.

PARAM NAME AND VALUE
The PARAM tag allows you to specify applet-specific arguments in an HTML page. Applets access their attributes with the getParameter( ) method.

6) Passing parameter to Applet

Example : abc.java

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=500 height=500>
                <param name="nm1" value=" Hello fp ">
        </applet>
*/
public class abc extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString(" getParameter() : "+getParameter("nm1"),10,10);
        }
}
```

7) Graphics class (java.awt.Graphics)

The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.
A Graphics object encapsulates state information needed for the basic rendering operations that Java supports. This state information includes the following properties:

- The Component object on which to draw.
- A translation origin for rendering and clipping coordinates.

- The current clip.
- The current color.
- The current font.

a) public abstract void drawLine(int x1,int y1,int x2,int y2)

- Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
- **Parameters: x1** - the first point's *x* coordinate.
  y1 - the first point's *y* coordinate.
  x2 - the second point's *x* coordinate.
  y2 - the second point's *y* coordinate.

Example – 1 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=500 height=500>
        </applet>
*/
public class abc extends Applet
{
        public void paint(Graphics g)
        {
                g.drawLine(10,20,100,200);
        }
}
```

b) public abstract void fillRect(int x,int y,int width,int height)

- Fills the specified rectangle. The left and right edges of the rectangle are at x and x + width - 1. The top and bottom edges are at y and y + height - 1. The resulting rectangle covers an area width pixels wide by height pixels tall. The rectangle is filled using the graphics context's current color.
- **Parameters: x** - the *x* coordinate of the rectangle to be filled.
  y - the *y* coordinate of the rectangle to be filled.
  width - the width of the rectangle to be filled.
  height - the height of the rectangle to be filled.

Example – 2 : abc.java

```
import java.awt.*;
import java.applet.*;

/*
        <applet code="abc" width=500 height=500>
        </applet>
```

```
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
		g.fillRect(10,20,100,200);
	}
}
```

c) public void drawRect(int x,int y,int width,int height)

- Draws the outline of the specified rectangle. The left and right edges of the rectangle are at x and x + width. The top and bottom edges are at y and y + height. The rectangle is drawn using the graphics context's current color.
- **Parameters: x** - the *x* coordinate of the rectangle to be drawn.
  y - the *y* coordinate of the rectangle to be drawn.
  width - the width of the rectangle to be drawn.
  height - the height of the rectangle to be drawn.

Example – 3 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
		g.drawRect(10,20,100,200);
	}
}
```

d) public abstract void drawRoundRect(int x,int y,int width,int height, int arcWidth, int arcHeight)

- Draws an outlined round-cornered rectangle using this graphics context's current color. The left and right edges of the rectangle are at x and x + width, respectively. The top and bottom edges of the rectangle are at y and y + height.
- **Parameters: x** - the *x* coordinate of the rectangle to be drawn.
  y - the *y* coordinate of the rectangle to be drawn.
  width - the width of the rectangle to be drawn.

height - the height of the rectangle to be drawn.
arcWidth - the horizontal diameter of the arc at the four corners.
arcHeight - the vertical diameter of the arc at the four corners.

Example – 4 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
		g.drawRoundRect(10,20,100,200,10,20);
	}
}
```

e) public abstract void fillRoundRect(int x,int y,int width,int height, int arcWidth, int arcHeight)

- Fills the specified rounded corner rectangle with the current color. The left and right edges of the rectangle are at x and x + width - 1, respectively. The top and bottom edges of the rectangle are at y and y + height - 1.
- **Parameters: x** - the *x* coordinate of the rectangle to be filled.
  y - the *y* coordinate of the rectangle to be filled.
  width - the width of the rectangle to be filled.
  height - the height of the rectangle to be filled.
  arcWidth - the horizontal diameter of the arc at the four corners.
  arcHeight - the vertical diameter of the arc at the four corners.

Example – 5 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
		g.fillRoundRect(10,20,100,200,10,20);
	}
```

```
}
```

f) public abstract void drawOval(int x,int y,int width,int height)

- Draws the outline of an oval. The result is a circle or ellipse that fits within the rectangle specified by the x, y, width, and height arguments.
- The oval covers an area that is width + 1 pixels wide and height + 1 pixels tall.
- **Parameters: x** - the *x* coordinate of the upper left corner of the oval to be drawn.
  y - the *y* coordinate of the upper left corner of the oval to be drawn.
  width - the width of the oval to be drawn.
  height - the height of the oval to be drawn.

Example – 6 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=500 height=500>
        </applet>
*/
public class abc extends Applet
{
        public void paint(Graphics g)
        {
                g.drawOval(10,20,100,200);
        }
}
```

g) public abstract void fillOval(int x,int y,int width,int height)

- Fills an oval bounded by the specified rectangle with the current color.
- **Parameters: x** - the *x* coordinate of the upper left corner of the oval to be filled.
  y - the *y* coordinate of the upper left corner of the oval to be filled.
  width - the width of the oval to be filled.
  height - the height of the oval to be filled.

Example – 7 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=500 height=500>
        </applet>
*/
public class abc extends Applet
{
        public void paint(Graphics g)
```

```
        {
                g.fillOval(10,20,100,200);
        }
}
```

h) public abstract void drawArc(int x,int y,int width,int height, int startAngle, int arcAngle)

- Draws the outline of a circular or elliptical arc covering the specified rectangle.
  Parameters: x - the *x* coordinate of the upper-left corner of the arc to be drawn.
  y - the *y* coordinate of the upper-left corner of the arc to be drawn.
  width - the width of the arc to be drawn.
  height - the height of the arc to be drawn.
  startAngle - the beginning angle.
  arcAngle - the angular extent of the arc, relative to the start angle.

Example – 8 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
        <applet code="abc" width=500 height=500>
        </applet>
*/
public class abc extends Applet
{
        public void paint(Graphics g)
        {
                g.drawArc(10,20,200,500,50,90);
        }
}
```

i) public abstract void fillArc(int x,int y,int width,int height, int startAngle, int arcAngle)

- Fills a circular or elliptical arc covering the specified rectangle
- **Parameters: x** - the *x* coordinate of the upper-left corner of the arc to be filled.
  y - the *y* coordinate of the upper-left corner of the arc to be filled.
  width - the width of the arc to be filled.
  height - the height of the arc to be filled.
  startAngle - the beginning angle.
  arcAngle - the angular extent of the arc, relative to the start angle.

Example – 9 : abc.java

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

```
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
		g.fillArc(10,20,200,500,50,90);
	}
}
```

j) public abstract void drawPolygon(int[] xPoints,int[] yPoints, int nPoints)

- Draws a closed polygon defined by arrays of *x* and *y* coordinates. Each pair of (*x*, *y*) coordinates defines a point.
- **Parameters: xPoints** - an array of x coordinates.
  yPoints - an array of y coordinates.
  nPoints - the total number of points.

Example – 10 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
		int i[] = {10,20,30,40};
		int j[] = {170,20,33,320};
		g.drawPolygon(i,j,4);
	}
}
```

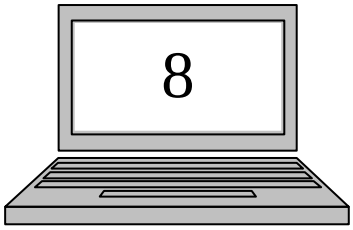
k) public abstract void drawString(String str,int x,int y)

- Draws the text given by the specified string, using this graphics context's current font and color. The baseline of the leftmost character is at position (*x*, *y*) in this graphics context's coordinate system.
- **Parameters: str** - the string to be drawn.
  x - the *x* coordinate.

y - the *y* coordinate.

Example – 11 : abc.java

```
import java.awt.*;
import java.applet.*;
/*
	<applet code="abc" width=500 height=500>
	</applet>
*/
public class abc extends Applet
{
	public void paint(Graphics g)
	{
		d.drawString(" Hello fp, How are you ? ");
	}
}
```



CHAPTER EIGHT

Swing

In This Chapter

- Swing

1) Swing

Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. Swing supplies several exciting additions, including tabbed panes, scroll panes, trees.

Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and, therefore, are platform-independent. The term lightweight is used to describe such elements.

a) Container class

The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container. This makes for a multileveled containment system. A container is responsible for laying out any components that it contains. It does this through the use of various layout managers.

b) JApplet class

This class is a container class and it contains other component on it.

Fundamental to Swing is the JApplet class, which extends Applet. Applets that use Swing must be subclasses of JApplet. JApplet is rich with functionality that is not found in Applet. For example, JApplet supports various “panes,” such as the content pane, the glass pane, and the root pane.

However, one difference between Applet and JApplet is that when adding a component to an instance of JApplet, do not invoke the add( ) method of the applet. Instead, call add( ) for the content pane of the JApplet object.

i) public void add(Component comp)

- Adds the specified component in this container.

ii) public Container getContentPane()

- Returns the contentPane object for this applet.

iii) public void remove(Component comp)

- Removes the specified component from the container.

iv) public void setLayout(LayoutManager manager)

- Sets the LayoutManager.

c) JPanel class

It is a lightweight container.

i) public JPanel()

- Creates a new JPanel with a double buffer and a flow layout.

ii) public JPanel(LayoutManager layout)

- Create a new buffered JPanel with the specified layout manager
- **Parameters: layout** - the LayoutManager to use

iii) public void add(Component comp)

- Adds the specified component in this container.

iv) public void remove(Component comp)

- Removes the specified component from the container

d) JFrame class

An extended version of java.awt.Frame that adds support for the JFC/Swing component architecture.

The JFrame class is slightly incompatible with Frame. Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The **content pane** provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame.

i) public JFrame() throws HeadlessException

- Constructs a new frame that is initially invisible.

ii) public JFrame(String title) throws HeadlessException

- Creates a new, initially invisible Frame with the specified title.

iii) public void setDefaultCloseOperation(int operation)

- Sets the operation that will happen by default when the user initiates a "close" on this frame.

- You must specify one of the following choices:
  DO_NOTHING_ON_CLOSE (defined in WindowConstants): Don't do anything; require the program to handle the operation in the windowClosing method of a registered WindowListener object.
  HIDE_ON_CLOSE (defined in WindowConstants): Automatically hide the frame after invoking any registered WindowListener objects.
  DISPOSE_ON_CLOSE (defined in WindowConstants): Automatically hide and dispose the frame after invoking any registered WindowListener objects.
  EXIT_ON_CLOSE (defined in JFrame): Exit the application using the System exit method. Use this only in applications.

The value is set to HIDE_ON_CLOSE by default.

iv) public void setJMenuBar(JMenuBar menubar)

- Sets the menubar for this frame.

v) public void remove(Component comp)

- Removes the specified component from the container.

vi) public void setLayout(LayoutManager manager)

- Sets the LayoutManager.

vii)public Container getContentPane()

- Returns the contentPane object for this frame.

viii) public void setTitle(String title)

- Sets the title for this frame to the specified string.
- **Parameters: title** - the title to be displayed in the frame's border. A null value is treated as an empty string, "".

ix) public String getTitle()

- Gets the title of the frame. The title is displayed in the frame's border.
- **Returns:** the title of this frame, or an empty string ("") if this frame doesn't have a title.

x) public void setBounds(int x,int y,int width,int height)

- Moves and resizes this component. The new location of the top-left corner is specified by x and y, and the new size is specified by width and height.
- **Parameters: x** - the new *x*-coordinate of this component
  y - the new *y*-coordinate of this component
  width - the new width of this component
  height - the new height of this component

xi) public void setVisible(boolean b)

- Shows or hides this Window depending on the value of parameter b.

xii)public void setEnabled(boolean b)

- Enables or disables this component, depending on the value of the parameter b. An enabled component can respond to user input and generate events. Components are enabled initially by default.

Example – 1 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	Ajf(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" JFrame demo ");
	}
}
```

e) JLabel class

A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

i) public JLabel()

- Creates a JLabel instance with no image and with an empty string for the title.

ii) public JLabel(Icon image)

- Creates a JLabel instance with the specified image. The label is centered vertically and horizontally in its display area.
- **Parameters: image** - The image to be displayed by the label.

iii) public JLabel(String text)

- Creates a JLabel instance with the specified text. The label is aligned against the leading edge of its display area, and centered vertically.
- **Parameters: text** - The text to be displayed by the label.

iv) public JLabel(String text,Icon icon,int horizontalAlignment)

- Creates a JLabel instance with the specified text, image, and horizontal alignment. The label is centered vertically in its display area. The text is on the trailing edge of the image.
- **Parameters: text** - The text to be displayed by the label.
  icon - The image to be displayed by the label.
  horizontalAlignment - One of the following constants defined in **SwingConstants**: LEFT, CENTER, RIGHT, LEADING or TRAILING.

v) public String getText()

- Returns the text string that the label displays.

vi) public void setText(String text)

- Defines the single line of text this component will display. If the value of text is null or empty string, nothing is displayed.

vii)public Icon getIcon()

- Returns the graphic image that the label displays.

viii) public void setIcon(Icon icon)

- Defines the icon this component will display. If the value of icon is null, nothing is displayed.

ix) public void setBounds(int x,int y,int width,int height)

- Moves and resizes this component. The new location of the top-left corner is specified by x and y, and the new size is specified by width and height.
- **Parameters: x** - the new *x*-coordinate of this component
  y - the new *y*-coordinate of this component
  width - the new width of this component
  height - the new height of this component

x) public void setVisible(boolean b)

- Shows or hides this Window depending on the value of parameter b.

xi) public void setEnabled(boolean b)

- Enables or disables this component, depending on the value of the parameter b. An enabled component can respond to user input and generate events. Components are enabled initially by default.

Example – 2 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
		JLabel lbl_msg = new JLabel(" Hello fp ");
		cp.add(lbl_msg);
		lbl_msg.setBounds(100,100,200,30);
		Icon i = new ImageIcon("1.png");
		JLabel lbl_img = new JLabel(i);
		cp.add(lbl_img);
		lbl_img.setBounds(100,200,100,100);
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" JLabel demo ");
	}
}
```

f) JTextField class

JTextField is a lightweight component that allows the editing of a single line of text.

i) public JTextField()

- Constructs a new TextField. A default model is created, the initial string is null, and the number of columns is set to 0.

ii) public JTextField(String text)

- Constructs a new TextField initialized with the specified text. A default model is created and the number of columns is 0.
- **Parameters: text** - the text to be displayed, or null

iii) public JTextField(int columns)

- Constructs a new empty TextField with the specified number of columns. A default model is created and the initial string is set to null.
- **Parameters: columns** - the number of columns to use to calculate the preferred width; if columns is set to zero, the preferred width will be whatever naturally results from the component implementation

iv) public JTextField(String text,int columns)

- Constructs a new TextField initialized with the specified text and columns. A default model is created.
- **Parameters: text** - the text to be displayed, or null
  columns - the number of columns to use to calculate the preferred width; if columns is set to zero, the preferred width will be whatever naturally results from the component implementation

v) public int getColumns()

- Returns the number of columns in this TextField.
- **Returns:** the number of columns >= 0

vi) public void setColumns(int columns)

- Sets the number of columns in this TextField, and then invalidate the layout.
- **Parameters: columns** - the number of columns >= 0

vii)public void setText(String text)

- Set the specified text in this JTextField.

viii) public String getText()

- Return the text of this JtextField.

Example – 3 : abc.java

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

```
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
		JTextField txt_msg = new JTextField(" Hello fp ");
		cp.add(txt_msg);
		txt_msg.setBounds(100,100,150,30);
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" JTextField demo ");
	}
}
```

g) JButton class

Use to create simple push button.

i) public JButton()

- Creates a button with no set text or icon

ii) public JButton(Icon icon)

- Creates a button with an icon.
- **Parameters: icon** - the Icon image to display on the button

iii) public JButton(String text)

- Creates a button with text.
- **Parameters: text** - the text of the button

iv) public JButton(String text,Icon icon)

- Creates a button with initial text and an icon.
- **Parameters: text** - the text of the button
  icon – the Icon image to display on the button

v) public String getText()

- Returns the text string that the JButton displays.

vi) public void setText(String text)

- Defines the single line of text this component will display. If the value of text is null or empty string, nothing is displayed.

vii)public Icon getIcon()

- Returns the graphic image that the JButton displays.

viii) public void setIcon(Icon icon)

- Defines the icon this component will display. If the value of icon is null, nothing is displayed.

ix) public void setBounds(int x,int y,int width,int height)

- Moves and resizes this component. The new location of the top-left corner is specified by x and y, and the new size is specified by width and height.
- **Parameters: x** - the new *x*-coordinate of this component
  y - the new *y*-coordinate of this component
  width - the new width of this component
  height - the new height of this component

x) public void setVisible(boolean b)

- Shows or hides this Window depending on the value of parameter b.

xi) public void setEnabled(boolean b)

- Enables or disables this component, depending on the value of the parameter b. An enabled component can respond to user input and generate events. Components are enabled initially by default.

Example – 4 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
        A(String tit)
        {
                super(tit);
                setVisible(true);
                setBounds(100,100,500,500);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
            Container cp = getContentPane();
            cp.setLayout(null);
            JButton b_msg = new JButton(" Click Me !!!! ");
            cp.add(b_msg);
            b_msg.setBounds(100,100,200,30);
            Icon i = new ImageIcon("1.png");
            JButton b_img = new JButton(i);
            cp.add(b_img);
            b_img.setBounds(100,200,100,100);
        }
}
class abc
{
        public static void main(String s[])
        {
            new A(" JButton demo ");
        }
}
```

h) JCheckBox class

An implementation of a check box -- an item that can be selected or deselected, and which displays its state to the user.

i) public JCheckBox()

- Creates an initially unselected check box button with no text, no icon.

ii) public JCheckBox(Icon icon)

- Creates an initially unselected check box with an icon.
- **Parameters: icon** - the Icon image to display

iii) public JCheckBox(Icon icon,boolean selected)

- Creates a check box with an icon and specifies whether or not it is initially selected.
- **Parameters: icon** - the Icon image to display
  selected - a boolean value indicating the initial selection state. If true the check box is selected

iv) public JCheckBox(String text)

- Creates an initially unselected check box with text.
- **Parameters: text** - the text of the check box.

v) public JCheckBox(String text,boolean selected)

- Creates a check box with text and specifies whether or not it is initially selected.
- **Parameters: text** - the text of the check box.
  selected - a boolean value indicating the initial selection state. If true the check box is selected

vi) public JCheckBox(String text,Icon icon)

- Creates an initially unselected check box with the specified text and icon.
- **Parameters: text** - the text of the check box.
  icon - the Icon image to display

vii)public String getText()

- Returns the text string that the JCheckBox displays.

viii) public void setText(String text)

- Defines the single line of text this component will display. If the value of text is null or empty string, nothing is displayed.

ix) public Icon getIcon()

- Returns the graphic image that the JCheckBox displays.

x) public void setIcon(Icon icon)

- Defines the icon this component will display. If the value of icon is null, nothing is displayed.

xi) public void setBounds(int x,int y,int width,int height)

- Moves and resizes this component. The new location of the top-left corner is specified by x and y, and the new size is specified by width and height.
- **Parameters: x** - the new *x*-coordinate of this component
  y - the new *y*-coordinate of this component
  width - the new width of this component
  height - the new height of this component

xii)public void setVisible(boolean b)

- Shows or hides this Window depending on the value of parameter b.

xiii) public void setEnabled(boolean b)

- Enables or disables this component, depending on the value of the parameter b. An enabled component can respond to user input and generate events. Components are enabled initially by default.

xiv) public Boolean isSelected()

- Test for this JCheckBox is selected or not.

Example – 5 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
		JCheckBox chk_msg = new JCheckBox(" Hello fp ",true);
		cp.add(chk_msg);
		chk_msg.setBounds(100,100,200,30);
		Icon i = new ImageIcon("1.png");
		JCheckBox chk_img = new JCheckBox(i);
		cp.add(chk_img);
		chk_img.setBounds(100,200,100,100);
		System.out.println(chk_msg.isSelected());
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" JCheckBox demo ");
	}
}
```

i) JRadioButton

An implementation of a radio button -- an item that can be selected or deselected, and which displays its state to the user. Used with a ButtonGroup object to create a group of buttons in which only one button at a time can be selected. (Create a ButtonGroup object and use its add method to include the JRadioButton objects in the group.)

The **ButtonGroup** class is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning "on" one of those buttons turns off all other buttons in the group.

A **ButtonGroup** can be used with any set of objects that inherit from AbstractButton. Typically a button group contains instances of JRadioButton.

i) public ButtonGroup()

- Creates a new ButtonGroup.

ii) public void add(AbstractButton b)

- Adds the button to the group.
- **Parameters: b** - the button to be added

iii) public int getButtonCount()

- Returns the number of buttons in the group.
- **Returns:** the button count

iv) public JRadioButton()

- Creates an initially unselected radio button with no set text.

v) public JRadioButton(Icon icon)

- Creates an initially unselected radio button with the specified image but no text.
  - **Parameters:** **icon** - the image that the button should display

vi) public JRadioButton(Icon icon,boolean selected)

- Creates a radio button with the specified image and selection state, but no text.
  - **Parameters: icon** - the image that the button should display
    selected - if true, the button is initially selected; otherwise, the button is initially unselected

vii)public JRadioButton(String text)

- Creates an unselected radio button with the specified text.
- **Parameters: text** - the string displayed on the radio button

viii) public JRadioButton(String text,boolean selected)

- Creates a radio button with the specified text and selection state.
- **Parameters: text** - the string displayed on the radio button
  selected - if true, the button is initially selected; otherwise, the button is initially unselected

ix) public JRadioButton(String text,Icon icon)

- Creates a radio button that has the specified text and image, and that is initially unselected.
- **Parameters: text** - the string displayed on the radio button
  icon - the image that the button should display

x) public JRadioButton(String text,Icon icon,boolean selected)

- Creates a radio button that has the specified text, image, and selection state.
  Parameters: text - the string displayed on the radio button
  icon - the image that the button should display
  selected – if true, the button is initially selected; otherwise, the button is initially unselected.

xi) public String getText()

- Returns the text string that the JRadioButton displays.

xii)public void setText(String text)

- Defines the single line of text this component will display. If the value of text is null or empty string, nothing is displayed.

xiii) public Icon getIcon()

- Returns the graphic image that the JRadionButton displays.

xiv) public void setIcon(Icon icon)

- Defines the icon this component will display. If the value of icon is null, nothing is displayed.

xv)public void setBounds(int x,int y,int width,int height)

- Moves and resizes this component. The new location of the top-left corner is specified by x and y, and the new size is specified by width and height.
- **Parameters: x** - the new *x*-coordinate of this component
  y - the new *y*-coordinate of this component
  width - the new width of this component
  height - the new height of this component

xvi) public void setVisible(boolean b)

- Shows or hides this Window depending on the value of parameter b.

xvii) public void setEnabled(boolean b)

- Enables or disables this component, depending on the value of the parameter b. An enabled component can respond to user input and generate events. Components are enabled initially by default.

xviii) public Boolean isSelected()

- Test for this JRadioButton is selected or not.

Example – 6 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
		JRadioButton rdo_msg1 = new JRadioButton(" RadioButton - 1 ");
		cp.add(rdo_msg1);
		rdo_msg1.setBounds(100,100,200,30);
		JRadioButton rdo_msg2 = new JRadioButton(" RadioButton - 2 ");
		cp.add(rdo_msg2);
		rdo_msg2.setBounds(100,150,200,30);
		JRadioButton rdo_msg3 = new JRadioButton(" RadioButton - 3 ",true);
		cp.add(rdo_msg3);
		rdo_msg3.setBounds(100,200,200,30);
		Icon i = new ImageIcon("1.png");
		JRadioButton rdo_img = new JRadioButton(i);
		cp.add(rdo_img);
		rdo_img.setBounds(100,250,100,100);
		ButtonGroup bg1 = new ButtonGroup();
		bg1.add(rdo_msg1);
		bg1.add(rdo_msg2);
		ButtonGroup bg2 = new ButtonGroup();
		bg2.add(rdo_msg3);
		bg2.add(rdo_img);
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" JRadioButton demo ");
```

```
    }
}
```

j) JComboBox class

A component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request. If you make the combo box editable, then the combo box includes an editable field into which the user can type a value.

i) public JComboBox()

- Creates a JComboBox with a default data model. The default data model is an empty list of objects. Use addItem to add items. By default the first item in the data model becomes selected.

ii) public JComboBox(Object[] items)

- Creates a JComboBox that contains the elements in the specified array. By default the first item in the array (and therefore the data model) becomes selected.

iii) public JComboBox(Vector<?> items)

- Creates a JComboBox that contains the elements in the specified Vector. By default the first item in the vector (and therefore the data model) becomes selected.

iv) public void setEditable(boolean aFlag)

- Determines whether the JComboBox field is editable. An editable JComboBox allows the user to type into the field or selected an item from the list to initialize the field, after which it can be edited.

v) public boolean isEditable()

- Returns true if the JComboBox is editable. By default, a combo box is not editable.
- **Returns:** true if the JComboBox is editable, else false

vi) public void setMaximumRowCount(int count)

- Sets the maximum number of rows the JComboBox displays. If the number of objects in the model is greater than count, the combo box uses a scrollbar.
- **Parameters: count** - an integer specifying the maximum number of items to display in the list before using a scrollbar

vii)public int getMaximumRowCount()

- Returns the maximum number of items the combo box can display without a scrollbar
- **Returns:** an integer specifying the maximum number of items that are displayed in the list before using a scrollbar

viii) public void setSelectedItem(Object anObject)

- Sets the selected item in the combo box display area to the object in the argument. If anObject is in the list, the display area shows anObject selected.

ix) public Object getSelectedItem()

- Returns the current selected item.

x) public void setSelectedIndex(int anIndex)

- Selects the item at index anIndex.

xi) public int getSelectedIndex()

- Returns the first item in the list that matches the given item. The result is not always defined if the JComboBox allows selected items that are not in the list. Returns -1 if there is no selected item or if the user specified an item which is not in the list.
- **Returns:** an integer specifying the currently selected list item, where 0 specifies the first item in the list; or -1 if no item is selected or if the currently selected item is not in the list

xii)public void addItem(Object anObject)

- Adds an item to the item list. This method works only if the JComboBox uses a mutable data model.

xiii) public void insertItemAt(Object anObject,int index)

- Inserts an item into the item list at a given index. This method works only if the JComboBox uses a mutable data model.
- **Parameters: anObject** - the Object to add to the list
  index - an integer specifying the position at which to add the item

xiv) public void removeItem(Object anObject)

- Removes an item from the item list. This method works only if the JComboBox uses a mutable data model.
- **Parameters: anObject** - the object to remove from the item list

xv)public void removeItemAt(int anIndex)

- Removes the item at anIndex This method works only if the JComboBox uses a mutable data model.

- **Parameters: anIndex** - an int specifying the index of the item to remove, where 0 indicates the first item in the list

xvi) public void removeAllItems()

- Removes all items from the item list.

xvii) public void setEnabled(boolean b)

- Enables the combo box so that items can be selected.

xviii) public int getItemCount()

- Returns the number of items in the list.
- **Returns:** an integer equal to the number of items in the list

xix) public Object getItemAt(int index)

- Returns the list item at the specified index. If index is out of range (less than zero or greater than or equal to size) it will return null.
- **Parameters: index** - an integer indicating the list position, where the first item starts at zero
- **Returns:** the Object at that list position; or null if out of range

Example – 7 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
		JComboBox cbo_sub = new JComboBox();
		cbo_sub.addItem(" C ");
		cbo_sub.addItem(" C++ ");
		cbo_sub.addItem(" JAVA ");
		cbo_sub.addItem(" J2EE ");
		cbo_sub.addItem(" J2ME ");
		cbo_sub.setEditable(true);
		cp.add(cbo_sub);
		cbo_sub.setBounds(100,100,150,30);
```

```
        }
}
class abc
{
        public static void main(String s[])
        {
                new A(" JComboBox demo ");
        }
}
```

k) JTextArea class

A JTextArea is a multi-line area that displays plain text.

i) public JTextArea()

- Constructs a new TextArea. A default model is set, the initial string is null, and rows/columns are set to 0.

ii) public JTextArea(String text)

- Constructs a new TextArea with the specified text displayed. A default model is created and rows/columns are set to 0.
- **Parameters: text** - the text to be displayed, or null

iii) public JTextArea(int rows,int columns)

- Constructs a new empty TextArea with the specified number of rows and columns. A default model is created, and the initial string is null.
- **Parameters: rows** - the number of rows >= 0
  columns - the number of columns >= 0

iv) public JTextArea(String text,int rows,int columns)

- Constructs a new TextArea with the specified text and number of rows and columns. A default model is created.
- **Parameters: text** - the text to be displayed, or null
  rows - the number of rows >= 0
  columns - the number of columns >= 0

v) public void setLineWrap(boolean wrap)

- Sets the line-wrapping policy of the text area.

vi) public boolean getLineWrap()

- Gets the line-wrapping policy of the text area.

vii)public int getLineCount()

- Determines the number of lines contained in the area.
- **Returns:** the number of lines > 0

viii) public void insert(String str,int pos)

- Inserts the specified text at the specified position.

ix) public void append(String str)

- Appends the given text to the end of the document

x) public void replaceRange(String str,int start,int end)

- Replaces text from the indicated start to end position with the new text specified.

xi) public int getRows()

- Returns the number of rows in the TextArea.
- **Returns:** the number of rows >= 0

xii)public int getColumns()

- Returns the number of columns in the TextArea.
- **Returns:** number of columns >= 0

Example – 8 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
        A(String tit)
        {
                super(tit);
                setVisible(true);
                setBounds(100,100,500,500);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                Container cp = getContentPane();
                cp.setLayout(null);
                JTextArea txt_msg = new JTextArea(" Hello fp, How are you ? ");
                cp.add(txt_msg);
                txt_msg.setBounds(100,100,200,100);
        }
}
class abc
{
```

```
    public static void main(String s[])
    {
        new A(" JFrame demo ");
    }
}
```

l) JList class

A component that displays a list of objects and allows the user to select one or more items.

i) public JList()

- Constructs a JList with an empty, read-only, model.

ii) public JList(Vector<?> listData)

- Constructs a JList that displays the elements in the specified Vector.

iii) public JList(Object[] listData)

- Constructs a JList that displays the elements in the specified array.

iv) public Color getSelectionForeground()

- Returns the color used to draw the foreground of selected items.

v) public void setSelectionForeground(Color selectionForeground)

- Sets the color used to draw the foreground of selected items,

vi) public Color getSelectionBackground()

- Returns the color used to draw the background of selected items.

vii)public void setSelectionBackground(Color selectionBackground)

- Sets the color used to draw the background of selected items

viii) public int getVisibleRowCount()

- Returns the value of the visibleRowCount property.

ix) public void setVisibleRowCount(int visibleRowCount)

- Sets the visibleRowCount property

x) public boolean isSelectedIndex(int index)

- Returns true if the specified index is selected, else false

xi) public int[] getSelectedIndices()

- Returns an array of all of the selected indices, in increasing order.
- **Returns:** all of the selected indices, in increasing order, or an empty array if nothing is selected

xii)public void setSelectedIndex(int index)

- Selects a single cell.

xiii) public void setSelectedIndices(int[] indices)

- Changes the selection to be the set of indices specified by the given array.

xiv) public Object[] getSelectedValues()

- Returns an array of all the selected values, in increasing order based on their indices in the list.
- **Returns:** the selected values, or an empty array if nothing is selected

xv)public int getSelectedIndex()

- Returns the smallest selected cell index; *the selection* when only a single item is selected in the list.

xvi) public Object getSelectedValue()

- Returns the value for the smallest selected cell index; *the selected value* when only a single item is selected in the list.

Example – 9 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
        A(String tit)
        {
                super(tit);
                setVisible(true);
                setBounds(100,100,500,500);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                Container cp = getContentPane();
                cp.setLayout(null);
                String sub[] = {" C " ," C++ "," JAVA "," J2EE "," J2ME "};
                JList lst = new JList(sub);
                cp.add(lst);
                lst.setBounds(100,100,100,100);
```

```
        }
}
class abc
{
        public static void main(String s[])
        {
                new A(" JList demo ");
        }
}
```

m) JScrollBar class

An implementation of a scrollbar. The user positions the knob in the scrollbar to determine the contents of the viewing area. The program typically adjusts the display so that the end of the scrollbar represents the end of the displayable contents, or 100% of the contents. The start of the scrollbar is the beginning of the displayable contents, or 0%. The position of the knob within those bounds then translates to the corresponding percentage of the displayable contents.

i) public JScrollBar()

- Creates a vertical scrollbar with the following initial values:
  minimum = 0
  maximum = 100
  value = 0
  extent = 10.

ii) public void setUnitIncrement(int unitIncrement)

- Sets the unitIncrement property.
- Note, that if the argument is equal to the value of Integer.MIN_VALUE, the most look and feels will not provide the scrolling to the right/down.

iii) public void setBlockIncrement(int blockIncrement)

- Sets the blockIncrement property.

iv) public int getValue()

- Returns the scrollbar's value.

v) public void setValue(int value)

- Sets the scrollbar's value..

vi) public int getMinimum()

- Returns the minimum value supported by the scrollbar (usually zero).

vii)public void setMinimum(int minimum)

- Sets the model's minimum property.

viii) public int getMaximum()

- The maximum value of the scrollbar is maximum - extent.

ix) Public void setMaximum(int maximum)

- Sets the model's maximum property.

x) public void setEnabled(boolean x)

- Enables the component so that the knob position can be changed. When the disabled, the knob position cannot be changed.

Example – 10 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
		JScrollBar jsb_v = new JScrollBar();
		cp.add(jsb_v);
		jsb_v.setBounds(100,100,30,250);
		JScrollBar jsb_h = new JScrollBar(JScrollBar.HORIZONTAL);
		cp.add(jsb_h);
		jsb_h.setBounds(150,100,250,30);
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" JScrollBar demo ");
	}
}
```

n) JMenuBar class

An implementation of a menu bar. You add JMenu objects to the menu bar to construct a menu. When the user selects a JMenu object, its associated JPopupMenu is displayed, allowing the user to select one of the JMenuItems on it.

i) public JMenuBar()

- Creates a new menu bar.

ii) public JMenu add(JMenu c)

- Appends the specified menu to the end of the menu bar.
- **Parameters: c** - the JMenu component to add
- **Returns:** the menu component

iii) public int getMenuCount()

- Returns the number of items in the menu bar.
- **Returns:** the number of items in the menu bar.

o) JMenu class

An implementation of a menu -- a popup window containing JMenuItems that is displayed when the user selects an item on the JMenuBar.

i) public JMenu()

- Constructs a new JMenu with no text.

ii) public JMenu(String s)

- Constructs a new JMenu with the supplied string as its text.
- **Parameters: s** - the text for the menu label

iii) public boolean isSelected()

- Returns true if the menu is currently selected (highlighted).

iv) public void setSelected(boolean b)

- Sets the selection status of the menu.

v) public JMenuItem add(JMenuItem menuItem)

- Appends a menu item to the end of this menu. Returns the menu item added.
- **Parameters: menuItem** - the JMenuitem to be added
- **Returns:** the JMenuItem added

vi) public void addSeparator()

- Appends a new separator to the end of the menu.

vii)public int getItemCount()

- Returns the number of items on the menu, including separators.

viii) public void remove(JMenuItem item)

- Removes the specified menu item from this menu. If there is no popup menu, this method will have no effect.
- **Parameters: item** - the JMenuItem to be removed from the menu

ix) public void removeAll()

- Removes all menu items from this menu.

p) JMenuItem class

An implementation of an item in a menu. A menu item is essentially a button sitting in a list. When the user selects the "button", the action associated with the menu item is performed.

i) public JMenuItem()

- Creates a JMenuItem with no set text or icon.

ii) public JMenuItem(String text)

- Creates a JMenuItem with the specified text.
- **Parameters: text** - the text of the JMenuItem

iii) public void setEnabled(boolean b)

- Enables or disables the menu item.

Example – 11 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
```

```
            cp.setLayout(null);
            JMenuBar mbar = new JMenuBar();
            setJMenuBar(mbar);
            JMenu sub = new JMenu(" Subject ");
            mbar.add(sub);
            JMenuItem c = new JMenuItem(" C ");
            sub.add(c);
            JMenuItem cpp = new JMenuItem(" C++ ");
            sub.add(cpp);
            sub.add(new JSeparator());
            JMenuItem java = new JMenuItem(" JAVA ");
            sub.add(java);
            JMenuItem j2ee = new JMenuItem(" J2EE ");
            sub.add(j2ee);
            JMenuItem j2me = new JMenuItem(" J2ME ");
            sub.add(j2me);
        }
}
class abc
{
        public static void main(String s[])
        {
            new A(" Menu demo ");
        }
}
```

q) JPasswordField class

JPasswordField is a lightweight component that allows the editing of a single line of text where the view indicates something was typed, but does not show the original characters.

i) public JPasswordField()

- Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.

ii) public JPasswordField(String text)

- Constructs a new JPasswordField initialized with the specified text. The document model is set to the default, and the number of columns to 0.

iii) public char getEchoChar()

- Returns the character to be used for echoing. The default is '*'.

iv) public void setEchoChar(char c)

- Sets the echo character for this JPasswordField.

v) public boolean echoCharIsSet()

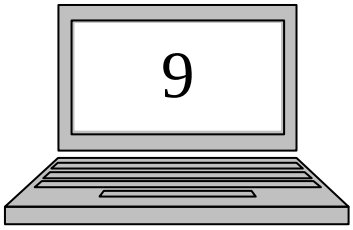
- Returns true if this JPasswordField has a character set for echoing.

vi) public char[] getPassword()

- Returns the text contained in this TextComponent.

Example – 12 : abc.java

```
import java.awt.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		setVisible(true);
		setBounds(100,100,500,500);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		Container cp = getContentPane();
		cp.setLayout(null);
		JPasswordField jpf = new JPasswordField();
		cp.add(jpf);
		jpf.setBounds(100,100,100,30);
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" JPasswordField demo ");
	}
}
```



CHAPTER NINE

LayoutManager

In This Chapter

- Layout Mangers

1) Layout Managers

A layout manager automatically arranges your controls within a window by using some type of algorithm. Each Container object has a layout manager associated with it. A layout manageris an instance of any class that implements the LayoutManager interface.

The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used. Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.

The setLayout( ) method has the following general form:
void setLayout(LayoutManager layoutObj)

If you wish to disable the layout manager and position components manually, pass null for layoutObj. If you do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component.

a) FlowLayout class

A flow layout arranges components in a directional flow, much like lines of text in a paragraph. The flow direction is determined by the container's componentOrientation property and may be one of two values:

- ComponentOrientation.LEFT_TO_RIGHT
- ComponentOrientation.RIGHT_TO_LEFT

The line alignment is determined by the align property. The possible values are:

- **public static final int LEFT**
  This value indicates that each row of components should be left-justified.
- **public static final int CENTER**
  This value indicates that each row of components should be centered.
- **public static final int RIGHT**
  This value indicates that each row of components should be right-justified.
- **public static final int LEADING**
  This value indicates that each row of components should be justified to the leading edge of the container's orientation, for example, to the left in left-to-right orientations.
- **public static final int TRAILING**

This value indicates that each row of components should be justified to the trailing edge of the container's orientation, for example, to the right in left-to-right orientations.

i) public FlowLayout()

- Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.

ii) public FlowLayout(int align)

- Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap. The value of the alignment argument must be one of FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER, FlowLayout.LEADING, or FlowLayout.TRAILING.
- **Parameters: align** - the alignment value

iii) public FlowLayout(int align,int hgap,int vgap)

- Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.
- The value of the alignment argument must be one of FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER, FlowLayout.LEADING, or FlowLayout.TRAILING.
- **Parameters: align** - the alignment value
  hgap - the horizontal gap between components and between the components and the borders of the Container
  vgap - the vertical gap between components and between the components and the borders of the Container

iv) public int getAlignment()

- Gets the alignment for this layout. Possible values are FlowLayout.LEFT, FlowLayout.RIGHT,FlowLayout.CENTER,FlowLayout.LEADING,FlowLayout.TRAILING.
- **Returns:** the alignment value for this layout

v) public void setAlignment(int align)

- Sets the alignment for this layout. Possible values are
- **Parameters: align** - one of the alignment values shown above

vi) public int getHgap()

- Gets the horizontal gap between components and between the components and the borders of the Container
- **Returns:** the horizontal gap between components and between the components and the borders of the Container

vii)public void setHgap(int hgap)

- Sets the horizontal gap between components and between the components and the borders of the Container.
- **Parameters: hgap** - the horizontal gap between components and between the components and the borders of the Container

viii) public int getVgap()

- Gets the vertical gap between components and between the components and the borders of the Container.
- **Returns:** the vertical gap between components and between the components and the borders of the Container

ix) public void setVgap(int vgap)

- Sets the vertical gap between components and between the components and the borders of the Container.
- **Parameters: vgap** - the vertical gap between components and between the components and the borders of the Container.

Example – 1 : abc.java

```
import java.awt.*;
import javax.swing.*;
class abc extends JFrame
{
	public static void main(String s[])
	{
		abc ft = new abc();
		ft.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		ft.setSize(400, 300);
		ft.setVisible(true);
	}
	abc()
	{
		super(" FlowLayout demo ");
		Container pane = getContentPane();
		pane.setLayout(new FlowLayout(FlowLayout.LEFT));
		pane.add(new JLabel("This is a test"));
		pane.add(new JButton("of a FlowLayout"));
		pane.add(new JTextField(30));
		pane.add(new JTextArea("This is a JTextArea", 3, 10));
	pane.add(new JLabel("This is a FlowLayout test with a long string"));
	}
```

}

b) BorderLayout class

A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER. When adding a component to a container with a border layout, use one of these five constants.

i) public BorderLayout()

- Constructs a new border layout with no gaps between components.

ii) public BorderLayout(int hgap,int vgap)

- Constructs a border layout with the specified gaps between components. The horizontal gap is specified by hgap and the vertical gap is specified by vgap.
- **Parameters: hgap** - the horizontal gap.
  vgap - the vertical gap.

iii) public int getHgap()

- Returns the horizontal gap between components

iv) public void setHgap(int hgap)

- Sets the horizontal gap between components.
- **Parameters:** hgap - the horizontal gap between components

v) public int getVgap()

- Returns the vertical gap between components.

vi) public void setVgap(int vgap)

- Sets the vertical gap between components.
- **Parameters: vgap** - the vertical gap between components

Example – 2 : abc.java

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.EtchedBorder;
class abc
{
        public static void main(String s[])
```

```
    {
        JFrame aWindow = new JFrame(" BorderLayout demo ");
        aWindow.setBounds(30, 30, 300, 300);
        aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        BorderLayout border = new BorderLayout();
        Container content = aWindow.getContentPane();
        content.setLayout(border);
        EtchedBorder edge = new EtchedBorder(EtchedBorder.RAISED);
        JButton button;
        content.add(button = new JButton("EAST"), BorderLayout.EAST);
        button.setBorder(edge);
        content.add(button = new JButton("WEST"), BorderLayout.WEST);
        button.setBorder(edge);
        content.add(button = new JButton("NORTH"), BorderLayout.NORTH);
        button.setBorder(edge);
        content.add(button = new JButton("SOUTH"), BorderLayout.SOUTH);
        button.setBorder(edge);
        content.add(button = new JButton("CENTER"), BorderLayout.CENTER);
        button.setBorder(edge);
        aWindow.setVisible(true);
    }
}
```

c) CardLayout class

A CardLayout object is a layout manager for a container. It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

i) public CardLayout()

- Creates a new card layout with gaps of size zero.

ii) public CardLayout(int hgap,int vgap)

- Creates a new card layout with the specified horizontal and vertical gaps. The horizontal gaps are placed at the left and right edges. The vertical gaps are placed at the top and bottom edges.
- **Parameters: hgap** - the horizontal gap.
  vgap - the vertical gap.

iii) public int getHgap()

- Gets the horizontal gap between components.

- **Returns:** the horizontal gap between components.

iv) public void setHgap(int hgap)
- Sets the horizontal gap between components.
- **Parameters: hgap** - the horizontal gap between components.

v) public int getVgap()
- Gets the vertical gap between components.
- **Returns:** the vertical gap between components.

vi) public void setVgap(int vgap)
- Sets the vertical gap between components.
- **Parameters: vgap** - the vertical gap between components.

vii)public void first(Container parent)
- Flips to the first card of the container.
- **Parameters: parent** - the parent container in which to do the layout

viii) public void next(Container parent)
- Flips to the next card of the specified container. If the currently visible card is the last one, this method flips to the first card in the layout.
- **Parameters: parent** - the parent container in which to do the layout

ix) public void previous(Container parent)
- Flips to the previous card of the specified container. If the currently visible card is the first one, this method flips to the last card in the layout.
- **Parameters: parent** - the parent container in which to do the layout

x) public void last(Container parent)
- Flips to the last card of the container.
- **Parameters: parent** - the parent container in which to do the layout

xi) public void show(Container parent,String name)
- Flips to the component that was added to this layout with the specified name, using addLayoutComponent. If no such component exists, then nothing happens.
- **Parameters: parent** - the parent container in which to do the layout
  name - the component name

Example – 3 : abc.java

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

```
class abc extends JFrame
{
	CardLayout layout;
	public static void main(String s[])
	{
		abc ct = new abc();
		ct.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		ct.displayTab("Blue Tab");
		ct.setSize(400, 300);
		ct.setVisible(true);
	}
	abc()
	{
		JPanel tab;
		Container pane = getContentPane();
		layout = new CardLayout();
		pane.setLayout(layout);
		tab = new JPanel();
		tab.setBackground(Color.red);
		pane.add(tab, "Red Tab");
		tab = new JPanel();
		tab.setBackground(Color.green);
		pane.add(tab, "Green Tab");
		tab = new JPanel();
		tab.setBackground(Color.blue);
		pane.add(tab, "Blue Tab");
	}
	void displayTab(String name)
	{
		layout.show(this.getContentPane(), name);
	}
}
```

Example – 4 : abc.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class abc extends JPanel implements ActionListener
{
	CardLayout card = new CardLayout(50, 50);
	abc()
	{
```

```
            setLayout(card);
            JButton button;
            for (int i = 1; i <= 6; i++)
            {
                  add(button = new JButton(" Press " + i), "Card" + i);
                  button.addActionListener(this);
            }
      }
      public void actionPerformed(ActionEvent e)
      {
            card.next(this);
      }
      public static void main(String[] args)
      {
            JFrame aWindow = new JFrame();
            aWindow.setBounds(30, 30, 300, 300);
            aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            aWindow.getContentPane().add(new abc());
            aWindow.setVisible(true);
      }
}
```

d) BoxLayout class

A layout manager that allows multiple components to be laid out either vertically or horizontally. The components will not wrap so, for example, a vertical arrangement of components will stay vertically arranged when the frame is resized.

Nesting multiple panels with different combinations of horizontal and vertical gives an effect similar to GridBagLayout, without the complexity. The BoxLayout manager is constructed with an axis parameter that specifies the type of layout that will be done.

There are four choices:

- **public static final int X_AXIS**

Specifies that components should be laid out left to right.

- **public static final int Y_AXIS**

Specifies that components should be laid out top to bottom.

- **public static final int LINE_AXIS**

Specifies that components should be laid out in the direction of a line of text as determined by the target container's ComponentOrientation property.

- **public static final int PAGE_AXIS**

Specifies that components should be laid out in the direction that lines flow across a page as determined by the target container's ComponentOrientation property.

i) public BoxLayout(Container target,int axis)

- Creates a layout manager that will lay out components along the given axis.
- **Parameters: target** - the container that needs to be laid out
  axis - the axis to lay out components along. Can be one of: BoxLayout.X_AXIS, BoxLayout.Y_AXIS, BoxLayout.LINE_AXIS or BoxLayout.PAGE_AXIS

ii) public final Container getTarget()

- Returns the container that uses this layout manager.
- **Returns:** the container that uses this layout manager

iii) public final int getAxis()

- Returns the axis that was used to lay out components. Returns one of: BoxLayout.X_AXIS, BoxLayout.Y_AXIS, BoxLayout.LINE_AXIS or BoxLayout.PAGE_AXIS
- **Returns:** the axis that was used to lay out components.

Example – 5 : abc.java

```
import java.awt.*;
import javax.swing.*;
class abc
{
	public static void main(String s[])
	{
		JFrame f = new JFrame(" BoxLayout demo");
		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Container pane = f.getContentPane();
		pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));
		for (float align = 0.0f; align <= 1.0f; align += 0.25f)
		{
			JButton button = new JButton("X Alignment = " + align);
			button.setAlignmentX(align);
			pane.add(button);
		}
		f.setSize(400, 300);
		f.setVisible(true);
	}
}
```

e) GridLayout class

The GridLayout class is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

i) public GridLayout()

- Creates a grid layout with a default of one column per component, in a single row.

ii) public GridLayout(int rows,int cols)

- Creates a grid layout with the specified number of rows and columns. All components in the layout are given equal size.
- One, but not both, of rows and cols can be zero, which means that any number of objects can be placed in a row or in a column.
- **Parameters: rows** - the rows, with the value zero meaning any number of rows.
  cols - the columns, with the value zero meaning any number of columns.

iii) public GridLayout(int rows,int cols,int hgap,int vgap)

- Creates a grid layout with the specified number of rows and columns. All components in the layout are given equal size.
- **Parameters: rows** - the rows, with the value zero meaning any number of rows
  cols - the columns, with the value zero meaning any number of columns
  hgap - the horizontal gap
  vgap - the vertical gap

iv) public int getRows()

- Gets the number of rows in this layout.
- **Returns:** the number of rows in this layout

v) public void setRows(int rows)

- Sets the number of rows in this layout to the specified value.
- **Parameters:** rows - the number of rows in this layout

vi) public int getColumns()

- Gets the number of columns in this layout.
- **Returns:** the number of columns in this layout

vii)public void setColumns(int cols)

- Sets the number of columns in this layout to the specified value.

viii) public int getHgap()

- Gets the horizontal gap between components.
- **Returns:** the horizontal gap between components

ix) public void setHgap(int hgap)

- Sets the horizontal gap between components to the specified value.
- **Parameters:** hgap - the horizontal gap between components

x) public int getVgap()

- Gets the vertical gap between components.
- **Returns:** the vertical gap between components

xi) public void setVgap(int vgap)

- Sets the vertical gap between components to the specified value.
- **Parameters: vgap** - the vertical gap between components

Example – 6 : abc.java

```
import java.awt.*;
import javax.swing.*;
class abc extends JFrame
{
	public static void main(String s[])
	{
		int rows = 2;
		int cols = 3;
		abc gt = new abc(rows, cols);
		gt.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		gt.pack();
		gt.setVisible(true);
	}
	public abc(int rows, int cols)
	{
		super(" GridLayout demo ");
		Container pane = getContentPane();
		pane.setLayout(new GridLayout(rows, cols));
		for (int i = 0; i < 20; i++)
		{
			JButton button = new JButton(Integer.toString(i + 1));
			pane.add(button);
		}
	}
}
```

f) GridBagLayout class

The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its *display area*.

Constant of this layout.

GridBagConstraints.NORTH
GridBagConstraints.SOUTH
GridBagConstraints.WEST
GridBagConstraints.EAST
GridBagConstraints.NORTHWEST
GridBagConstraints.NORTHEAST
GridBagConstraints.SOUTHWEST
GridBagConstraints.SOUTHEAST
GridBagConstraints.CENTER

i) public GridBagLayout()

- Creates a grid bag layout manager.

ii) public void setConstraints(Component comp, GridBagConstraints constraints)

- Sets the constraints for the specified component in this layout.
- **Parameters: comp** - the component to be modified
  constraints - the constraints to be applied

iii) public GridBagConstraints getConstraints(Component comp)

- Gets the constraints for the specified component. A copy of the actual GridBagConstraints object is returned.
- **Parameters: comp** - the component to be queried
- **Returns:** the constraint for the specified component in this grid bag layout; a copy of the actual constraint object is returned

Example – 7 : abc.java

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
class abc
{
        static JFrame aWindow = new JFrame("GridbagLayout demo");
        public static void main(String s[])
```

```
    {
        aWindow.setBounds(30, 30, 300, 300);
        aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints constraints = new GridBagConstraints();
        aWindow.getContentPane().setLayout(gridbag);
        constraints.weightx = constraints.weighty = 10.0;
        constraints.fill = constraints.BOTH;
        addButton(" Press ", constraints, gridbag);
        constraints.gridwidth = constraints.REMAINDER;
        addButton("GO", constraints, gridbag);
        aWindow.setVisible(true);
    }
    static void addButton(String label, GridBagConstraints constraints,
    GridBagLayout layout)
    {
        Border edge = BorderFactory.createRaisedBevelBorder();
        JButton button = new JButton(label);
        button.setBorder(edge);
        layout.setConstraints(button, constraints);
        aWindow.getContentPane().add(button);
    }
}
```

g) GroupLayout class

GroupLayout is a LayoutManager that hierarchically groups components in order to position them in a Container. GroupLayout is intended for use by builders, but may be hand-coded as well. Grouping is done by instances of the Group class. GroupLayout supports two types of groups. A sequential group positions its child elements sequentially, one after another. A parallel group aligns its child elements in pararlly.

i) public GroupLayout(Container host)

- Creates a GroupLayout for the specified Container.
- **Parameters: host** - the Container the GroupLayout is the LayoutManager for

ii) public void setAutoCreateGaps(boolean autoCreatePadding)

- Sets whether a gap between components should automatically be created.

iii) public boolean getAutoCreateGaps()

- Returns true if gaps between components are automatically created.
- **Returns:** true if gaps between components are automatically created

iv) public GroupLayout.SequentialGroup createSequentialGroup()

- Creates and returns a SequentialGroup.
- **Returns:** a new SequentialGroup

v) public GroupLayout.ParallelGroup createParallelGroup()

- Creates and returns a ParallelGroup with an alignment of Alignment.LEADING. This is a cover method for the more general createParallelGroup(Alignment) method.
- **Returns:** a new ParallelGroup.

vi) publicvoidreplace(Component existingComponent,Component new Component)

- Replaces an existing component with a new one.
- **Parameters: existingComponent** - the component that should be removed and replaced with newComponent

  newComponent - the component to put in existingComponent's place

vii)public void addLayoutComponent(String name, Component component)

- Notification that a Component has been added to the parent container. You should not invoke this method directly, instead you should use one of the Group methods to add a Component.

Example – 8 : abc.java

```
import javax.swing.*;
class abc
{
	public static void main(String s[])
	{
		JFrame frame = new JFrame(" GroupLayout demo");
		JPanel panel = new JPanel();
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		GroupLayout layout = new GroupLayout(panel);
		panel.setLayout(layout);
		JButton buttonD = new JButton("D");
		JButton buttonR = new JButton("R");
		JButton buttonY = new JButton("Y");
		JButton buttonO = new JButton("O");
		JButton buttonT = new JButton("T");
		GroupLayout.SequentialGroup leftToRight =
		layout.createSequentialGroup();
		GroupLayout.ParallelGroup columnMiddle =
		layout.createParallelGroup();
```

```
            columnMiddle.addComponent(buttonR);
            columnMiddle.addComponent(buttonO);
            columnMiddle.addComponent(buttonT);
            leftToRight.addGroup(columnMiddle);
            leftToRight.addComponent(buttonY);
            leftToRight.addComponent(buttonD);
            GroupLayout.SequentialGroup topToBottom =
            layout.createSequentialGroup();
            GroupLayout.ParallelGroup rowTop =
            layout.createParallelGroup();
            rowTop.addComponent(buttonD);
            rowTop.addComponent(buttonR);
            rowTop.addComponent(buttonY);
            topToBottom.addGroup(rowTop);
            topToBottom.addComponent(buttonO);
            topToBottom.addComponent(buttonT);
            layout.setHorizontalGroup(leftToRight);
            layout.setVerticalGroup(topToBottom);
            frame.add(panel);
            frame.pack();
            frame.setVisible(true);
        }
}
```

h) SpringLayout class

A SpringLayout lays out the children of its associated container according to a set of constraints.

Each constraint, represented by a Spring object, controls the vertical or horizontal distance between two component edges. The edges can belong to any child of the container, or to the container itself. For example, the allowable width of a component can be expressed using a constraint that controls the distance between the west (left) and east (right) edges of the component.

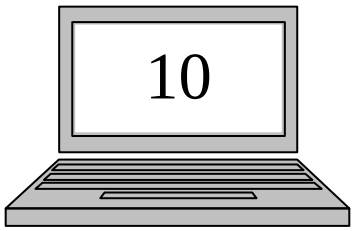
i) public SpringLayout()

- Constructs a new SpringLayout.

Example – 9 : abc.java

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

```
{
	public static void main(String args[])
	{
		JFrame frame = new JFrame("SpringLayout demo");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Container contentPane = frame.getContentPane();
		SpringLayout layout = new SpringLayout();
		contentPane.setLayout(layout);
		Component left = new JLabel("Left");
		Component right = new JTextField(15);
		contentPane.add(left);
		contentPane.add(right);
		layout.putConstraint(SpringLayout.WEST, left, 10,
		SpringLayout.WEST, contentPane);
		layout.putConstraint(SpringLayout.NORTH, left, 25,
		SpringLayout.NORTH, contentPane);
		layout.putConstraint(SpringLayout.NORTH, right, 25,
		SpringLayout.NORTH, contentPane);
		layout.putConstraint(SpringLayout.WEST, right, 20,
		SpringLayout.EAST, left);
		frame.setSize(300, 100);
		frame.setVisible(true);
	}
}
```



CHAPTER TEN

Event Handling

In This Chapter

- Event Handling
- Addapter Classes

1)Event Handling

a) Delegation Event Model

Its concept is quite simple: a source generates an event and sends it to one or more listeners.In this process the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to “delegate” the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.

Events

In the delegation model, an event is an object that describes a state change in a source. Some of the example of Events are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.

Event Sources

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Here is the general form:

public void addTypeListener(TypeListener el)

Here, Type is the name of the event and el is a reference to the event listener.

A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:

public void removeTypeListener(TypeListener el)

Here, Type is the name of the event and el is a reference to the event listener.

Event Listeners

A listener is an object that is notified when an event occurs. It has two major requirements First, it must have been registered with one or more sources to receive

specific types of events. Second, it must implement methods to receive and process these events.

b) ActionEvent class

An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

The ActionEvent class defines four integer constants that can be used to identify any modifiers associated with an action event:

- **public static final int ALT_MASK**

The alt modifier. An indicator that the alt key was held down during the event.

- **public static final int SHIFT_MASK**

The shift modifier. An indicator that the shift key was held down during the event.

- **public static final int CTRL_MASK**

The control modifier. An indicator that the control key was held down during the event.

- **public static final int META_MASK**

The meta modifier. An indicator that the meta key was held down during the event.

i) public ActionEvent(Object source,int id,String command)

- Constructs an ActionEvent object.
- **Parameters: source** - the object that originated the event
  Id - an integer that identifies the event
  command - a string that may specify a command (possibly one of several)associated with the event

ii) public ActionEvent(Object source,int id,String command,int modifiers)

- Constructs an ActionEvent object with modifier keys.
- **Parameters: source** - the object that originated the event
  id - an integer that identifies the event
  command - a string that may specify a command (possibly one of several) associated with the event
  modifiers - the modifier keys held down during this action

iii) publicActionEvent(Object source,int id,String command,long when, int modifiers)

- Constructs an ActionEvent object with the specified modifier keys and timestamp.
- **Parameters: source** - the object that originated the event
  id - an integer that identifies the event
  command - a string that may specify a command (possibly one

of several) associated with the event
when - the time the event occurred
modifiers - the modifier keys held down during this action

iv) public String getActionCommand()

- Returns the command string associated with this action.

v) public long getWhen()

- Returns the timestamp of when this event occurred.

c) ActionListener interface

The listener interface for receiving action events.

i) void actionPerformed(ActionEvent e)

- Envoked when an action event occurs.

Example – 1 : abc.java

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class abc extends JFrame implements ActionListener
{
	JButton bmsg1,bmsg2;
	JLabel lbl_disp;
	abc(String tit)
	{
		super(tit);
		Container cp = getContentPane();
		cp.setLayout(null);
		setBounds(100,100,500,500);
		setVisible(true);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		lbl_disp = new JLabel(" Click one of this 2 Button" );
		cp.add(lbl_disp);
		lbl_disp.setBounds(100,100,250,30);
		bmsg1 = new JButton(" Message 1 ");
		cp.add(bmsg1);
		bmsg1.setBounds(150,200,150,30);
		bmsg2 = new JButton(" Message 2 ");
		cp.add(bmsg2);
		bmsg2.setBounds(300,200,150,30);
		bmsg1.addActionListener(this);
```

```
            bmsg2.addActionListener(this);
        }

public void actionPerformed(ActionEvent ae)
        {
            lbl_disp.setText(ae.getActionCommand());
        }
        public static void main(String s[])
        {
            new abc(" ActionEvent demo ");
        }
}
```

d) AdjustmentEvent class

Occurs when scroll bar's value changed.

This class has following 4 constant.

- **BLOCK_DECREMENT**

The user clicked inside the scroll bar to decrease its value.

- **BLOCK_INCREMENT**

The user clicked inside the scroll bar to increase ts value.

- **TRACK**

The slider was dragged.

- **UNIT_DECREMENT**

The button at the end of the scroll bar was clicked to decrease its value.

- **UNIT_INCREMENT**

The button at the end of the scroll bar was clicked to increase its value.

i) public AdjustmentEvent(Adjustable source,int id,int type,int value)

- Constructs an AdjustmentEvent object with the specified Adjustable source, event type, adjustment type, and value.
- **Parameters: source** - the Adjustable object where the event originated
  id - the event type
  type - the adjustment type
  value - the current value of the adjustment

ii) public Adjustable getAdjustable()

- Returns the Adjustable object where this event originated.
- **Returns:** the Adjustable object where this event originated

iii) public int getValue()

- Returns the current value in the adjustment event.
- **Returns:** the current value in the adjustment event

iv) public int getAdjustmentType()

- Returns the type of adjustment which caused the value changed event.

e) AdjustmentListener Interface

The listener interface for receiving adjustment events.

i) void adjustmentValueChanged(AdjustmentEvent e)

- Invoked when the value of the adjustable has changed.

Example – 2 : abc.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class abc extends JFrame implements AdjustmentListener
{
        JScrollBar bar;
        abcade()
        {
                super(“ AdjustmentEvent demo “);
                bar = new JScrollBar(SwingConstants.HORIZONTAL, 50, 10, 0, 100);
                setSize(350, 100);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                bar.addAdjustmentListener(this);
                JPanel pane = new JPanel();
                pane.setLayout(new BorderLayout());
                pane.add(bar, "South");
                setContentPane(pane);
        }
        public static void main(String s[])
        {
                JFrame frame = new abcade();
                frame.setVisible(true);
        }
        public void adjustmentValueChanged(AdjustmentEvent evt)
        {
                Object source = evt.getSource();
                int newValue = bar.getValue();
```

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

f) ComponentEvent class

A low-level event which indicates that a component moved, changed size, or changed visibility

This class has following 4 constants.

- **public static final int COMPONENT_MOVED**

This event indicates that the component's position changed.

- **public static final int COMPONENT_RESIZED**

This event indicates that the component's size changed.

- **public static final int COMPONENT_SHOWN**

This event indicates that the component was made visible.

- **public static final int COMPONENT_HIDDEN**

This event indicates that the component was rendered invisible.

i) public ComponentEvent(Component source,int id)

- Constructs a ComponentEvent object.
- **Parameters: source** - the Component that originated the event
  id - an integer indicating the type of event

ii) public Component getComponent()

- Returns the originator of the event.
- **Returns:** the Component object that originated the event, or null if the object is not a Component.

g) ComponentLIstener interface

The listener interface for receiving component events.

i) void componentResized(ComponentEvent e)

- Invoked when the component's size changes.

ii) void componentMoved(ComponentEvent e)

- Invoked when the component's position changes

iii) void componentShown(ComponentEvent e)

- Invoked when the component has been made visible.

iv) void componentHidden(ComponentEvent e)

- Invoked when the component has been made invisible.

Example – 3 : abc.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class abc
{
	public static void main(String s[])
	{
		JFrame frame = new JFrame(" ComponentEvent demo ");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JCheckBox checkbox = new JCheckBox("Label visible", true);
		checkbox.addComponentListener(new ComponentListener()
		{
			public void componentHidden(ComponentEvent e)
			{
			System.out.println("componentHidden event from " +
		e.getComponent().getClass().getName());
			}
			public void componentMoved(ComponentEvent e)
			{
			   Component c = e.getComponent();
System.out.println("componentMoved event from " + c.getClass().getName() + "; new
location: "+c.getLocation().x + ", " + c.getLocation().y);
			}
			public void componentResized(ComponentEvent e)
			{
				Component c = e.getComponent();
System.out.println("componentResized event from " + c.getClass().getName() + "; new
size: "+c.getSize().width + ", " + c.getSize().height);
			}
			public void componentShown(ComponentEvent e)
			{
System.out.println("componentShown event from " +
e.getComponent().getClass().getName());
			}
		});
		frame.add(checkbox, "North");
		frame.setSize(300, 200);
		frame.setVisible(true);
```

```
    }
}
```

h) ContainerEvent class

A low-level event which indicates that a container's contents changed because a component was added or removed

This class has following 2 constants.

- **public static final int COMPONENT_ADDED**

This event indicates that a component was added to the container.

- **public static final int COMPONENT_REMOVED**

This event indicates that a component was removed from the container.

i) public ContainerEvent(Component source,int id,Component child)

- Constructs a ContainerEvent object.
- **Parameters: source** - the Component object (container) that originated the event
  id - an integer indicating the type of event
  child - the component that was added or removed

ii) public Container getContainer()

- Returns the originator of the event.
- **Returns:** the Container object that originated the event, or null if the object is not a Container.

iii) public Component getChild()

- Returns the component that was affected by the event.
- **Returns:** the Component object that was added or removed

i) ContainerListener interface

The listener interface for receiving container events.

i) void componentAdded(ContainerEvent e)

- Invoked when a component has been added to the container.

ii) void componentRemoved(ContainerEvent e)

- Invoked when a component has been removed from the container.

Example – 4 : abc.java

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

```
import javax.swing.*;
class abc
{
	public static void main(String s[])
	{
		JFrame frame = new JFrame(" ContainerEvent demo ");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JPanel buttonPanel = new JPanel();
		buttonPanel.addContainerListener(new ContainerListener()
		{
			public void componentAdded(ContainerEvent e)
			{
				displayMessage(" added to ", e);
			}
			public void componentRemoved(ContainerEvent e)
			{
				displayMessage(" removed from ", e);
			}
			void displayMessage(String action, ContainerEvent e)
			{
System.out.println(((JButton) e.getChild()).getText() + " was" +
action+e.getContainer().getClass().getName());
			}
		});
		buttonPanel.add(new JButton("A"));
		frame.add(buttonPanel);
		frame.setSize(300, 200);
		frame.setVisible(true);
	}
}
```

j) FocusEvent class

A low-level event which indicates that a Component has gained or lost the input focus.

This class has following 2 constants.

- **public static final int FOCUS_GAINED**

This event indicates that the Component is now the focus owner.

- **public static final int FOCUS_LOST**

This event indicates that the Component is no longer the focus owner.

i) PublicFocusEvent(Component source,int id,boolean temporary, Component opposite)

- Constructs a FocusEvent object with the specified temporary state and opposite Component. The opposite Component is the other Component involved in this focus change. For a FOCUS_GAINED event, this is the Component that lost focus. For a FOCUS_LOST event, this is the Component that gained focus.
- **Parameters: source** - the Component that originated the event
  id - FOCUS_GAINED or FOCUS_LOST
  temporary - true if the focus change is temporary; false otherwise
  opposite - the other Component involved in the focus change, or null

ii) public FocusEvent(Component source,int id,boolean temporary)

- Constructs a FocusEvent object and identifies whether or not the change is temporary.
- **Parameters: source** - the Component that originated the event
  id - an integer indicating the type of event
  temporary - true if the focus change is temporary; false otherwise

iii) public FocusEvent(Component source,int id)

- Constructs a FocusEvent object and identifies it as a permanent change in focus.
  Parameters: source - the Component that originated the event
  id - an integer indicating the type of event

iv) public boolean isTemporary()

- Identifies the focus change event as temporary or permanent.
- **Returns:** true if the focus change is temporary; false otherwise

v) public Component getOppositeComponent()

- Returns the other Component involved in this focus change.

k) FocusListener interface

The listener interface for receiving keyboard focus events on a component.

i) void focusGained(FocusEvent e)

- Invoked when a component gains the keyboard focus.

ii) void focusLost(FocusEvent e)

- Invoked when a component loses the keyboard focus.

Example – 5 : abc.java

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

```
import javax.swing.*;
class abc
{
	public static void main(String s[])
	{
		JFrame frame = new JFrame(" FocusEvent demo ");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JTextField textField = new JTextField("A TextField");
		textField.addFocusListener(new FocusListener()
		{
			public void focusGained(FocusEvent e)
			{
				displayMessage("Focus gained", e);
			}
			public void focusLost(FocusEvent e)
			{
				displayMessage("Focus lost", e);
			}
			void displayMessage(String prefix, FocusEvent e)
			{
System.out.println(prefix + (e.isTemporary() ? " (temporary):" : ":") + e.getComponent().getClass().getName() + "; Opposite component: " + (e.getOppositeComponent() != null ? e.getOppositeComponent().getClass().getName() : "null"));
			}
		});
		frame.add(textField,"North");
		frame.add(new JTextField(),"South");
		frame.setSize(300, 200);
		frame.setVisible(true);
	}
}
```

l) InputEvent class

InputEvent defines several integer constants that represent any modifiers, such as the control key being pressed, that might be associated with the event. Originally, the InputEvent class defined the following eight values to represent the modifiers.

CTRL_MASK, ALT_MASK, SHIFT_MASK, META_MASK, ALT_GRAPH_MASK, BUTTON1_MASK, BUTTON2_MASK, BUTTON3_MASK

However, because of possible conflicts between the modifiers used by keyboard events and mouse events, and other issues, Java 2, version 1.4 added the following extended modifier values.

CTRL_DOWN_MASK,ALT_DOWN_MASK,SHIFT_DOWN_MASK,META_DOWN_MASK, ALT_GRAPH_DOWN_MASK,BUTTON1_DOWN_MASK,BUTTON2_DOWN_MASK,BUTTON3_DOWN_MASK

When writing new code, it is recommended that you use the new, extended modifiers rather than the original modifiers.

i) public boolean isShiftDown()

- Returns whether or not the Shift modifier is down on this event.

ii) public boolean isControlDown()

- Returns whether or not the Control modifier is down on this event.

iii) public boolean isMetaDown()

- Returns whether or not the Meta modifier is down on this event.

iv) public boolean isAltDown()

- Returns whether or not the Alt modifier is down on this event.

v) public long getWhen()

- Returns the timestamp of when this event occurred.

vi) public int getModifiers()

- Returns the modifier mask for this event.

vii)public int getModifiersEx()

- Returns the extended modifier mask for this event. Extended modifiers represent the state of all modal keys, such as ALT, CTRL, META, and the mouse buttons just after the event occurred

m) ItemEvent class

A semantic event which indicates that an item was selected or deselected. This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user.

This class has following 2 constants.

- **public static final int SELECTED**

This state-change value indicates that an item was selected.

- **public static final int DESELECTED**

This state-change-value indicates that a selected item was deselected

i) public ItemEvent(ItemSelectable source,int id,Object item,int stateChange)

- Constructs an ItemEvent object.
- **Parameters: source** - the ItemSelectable object that originated the event
  id - an integer that identifies the event type
  item - an object -- the item affected by the event
  stateChange - an integer that indicates whether the item was selected or deselected

ii) public ItemSelectable getItemSelectable()

- Returns the originator of the event.
- **Returns:** the ItemSelectable object that originated the event.

iii) public Object getItem()

- Returns the item affected by the event.
- **Returns:** the item (object) that was affected by the event

iv) public int getStateChange()

- Returns the type of state change (selected or deselected).
- **Returns:** an integer that indicates whether the item was selected or deselected

n) ItemListener interface

The listener interface for receiving item events.

i) void itemStateChanged(ItemEvent e)

- Invoked when an item has been selected or deselected by the user. The code written for this method performs the operations that need to occur when an item is selected (or deselected).

Example – 6 : abc.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class abc
{
	public static void main(String s[])
	{
		JFrame frame = new JFrame("ItemEvent demo");
```

```
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            JCheckBox aCheckBox = new JCheckBox("Check it");
            ItemListener itemListener = new ItemListener()
            {
            public void itemStateChanged(ItemEvent itemEvent)
                  {
            AbstractButton abstractButton = (AbstractButton)itemEvent.getSource();
                  Color foreground = abstractButton.getForeground();
                        Color background = abstractButton.getBackground();
                        int state = itemEvent.getStateChange();
                        if (state == ItemEvent.SELECTED)
                        {
                              abstractButton.setForeground(background);
                              abstractButton.setBackground(foreground);
                        }
                  }
            };
            aCheckBox.addItemListener(itemListener);
            frame.add(aCheckBox);
            frame.setSize(300, 200);
            frame.setVisible(true);
      }
}
```

o) KeyEvent class

An event which indicates that a keystroke occurred in a component.

This class has following constant.

- **public static final int KEY_PRESSED**

The "key pressed" event. This event is generated when a key is pushed down.

- **public static final int KEY_RELEASED**

The "key released" event. This event is generated when a key is let up.

- **public static final int KEY_TYPED**

The "key typed" event. This event is generated when a character is entered. In the simplest case, it is produced by a single key press. Often, however, characters are produced by series of key presses, and the mapping from key pressed events to key typed events may be many-to-one or many-to-many.

There are many other integer constants that are defined by KeyEvent. For example

VK_0 to VK_9

VK_A to VK_Z define the ASCII equivalents of the numbers and letters.

Here are some others:

VK_ENTER, VK_ESCAPE, VK_CANCEL, VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_PAGE_DOWN,VK_PAGE_UP, VK_SHIFT, VK_ALT, VK_CONTROL

The VK constants specify virtual key codes and are independent of any modifiers, such as control, shift, or alt.

i) public KeyEvent(Component source,int id,long when,int modifiers, int keyCode,char keyChar)

- Constructs a KeyEvent object.
- **Parameters: source** - the Component that originated the event
  id - an integer identifying the type of event
  when - a long integer that specifies the time the event occurred
  modifiers - the modifier keys down during event (shift, ctrl, alt, meta) Either extended _DOWN_MASK or old _MASK modifiers should be used, but both models should not be mixed in one event. Use of the extended modifiers is preferred.
  keyCode - the integer code for an actual key, or VK_UNDEFINED (for a key- typed event)
  keyChar - the Unicode character generated by this event, or CHAR_UNDEFINED (for key-pressed and key-released events which do not map to a valid Unicode character)

ii) public int getKeyCode()

- Returns the integer keyCode associated with the key in this event.
- **Returns:** the integer code for an actual key on the keyboard. (For KEY_TYPED events, the keyCode is VK_UNDEFINED.)

iii) public char getKeyChar()

- Returns the character associated with the key in this event. For example, the KEY_TYPED event for shift + "a" returns the value for "A".

p) KeyListener interface

The listener interface for receiving keyboard events (keystrokes).

i) void keyTyped(KeyEvent e)

- Invoked when a key has been typed. See the class description for KeyEvent for a definition of a key typed event.

ii) void keyPressed(KeyEvent e)

- Invoked when a key has been pressed. See the class description for KeyEvent for a definition of a key pressed event.

iii) void keyReleased(KeyEvent e)

- Invoked when a key has been released. See the class description for KeyEvent for a definition of a key released event.

Example – 7 : abc.java

```
import java.awt.event.*;
import javax.swing.*;
class A implements KeyListener
{
	public void keyTyped(KeyEvent e)
	{
		char c = e.getKeyChar();
		System.out.println("Key Typed: " + c);
	}
	public void keyPressed(KeyEvent e)
	{
		char c = e.getKeyChar();
		System.out.println("Key Pressed: " + c);
	}
	public void keyReleased(KeyEvent e)
	{
		char c = e.getKeyChar();
		System.out.println("Key Released: " + c);
	}
}
class abc
{
	public static void main(String s[])
	{
		JFrame frame = new JFrame("KeyEvent demo");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JTextField textField = new JTextField();
		textField.addKeyListener(new A());
		frame.add(textField);
		frame.setSize(300, 200);
		frame.setVisible(true);
	}
```

}

q) MouseEvent class

An event which indicates that a mouse action occurred in a component.

This class has following constants.

- **MOUSE_CLICKED**

The user clicked the mouse.

- **MOUSE_DRAGGED**

The user dragged the mouse.

- **MOUSE_ENTERED**

The mouse entered a component.

- **MOUSE_EXITED**

The mouse exited from a component.

- **MOUSE_MOVED**

The mouse moved.

- **MOUSE_PRESSED**

The mouse was pressed.

- **MOUSE_RELEASED**

The mouse was released.

- **MOUSE_WHEEL**

The mouse wheel was moved.

- **NOBUTTON**

Indicates no mouse buttons;

- **BUTTON1**

Indicates mouse button #1;

- **BUTTON2**

Indicates mouse button #2

- **BUTTON3**

Indicates mouse button #3;

i) Public MouseEvent(Component source,int id,long when,int modifiers,int x, int y, int xAbs,int yAbs,int clickCount,boolean popupTrigger, int button)

- Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, absolute coordinates, and click count.
- **Parameters: source** - the Component that originated the event
  id - the integer that identifies the event
  when - a long int that gives the time the event occurred
  modifiers - the modifier keys down during event (e.g. shift, ctrl, alt, meta)
  Either extended _DOWN_MASK or old _MASK modifiers should be used,

but both models should not be mixed in one event. Use of the extended modifiers is preferred.
x - the horizontal x coordinate for the mouse location
y - the vertical y coordinate for the mouse location
xAbs - the absolute horizontal x coordinate for the mouse location
yAbs - the absolute vertical y coordinate for the mouse location
clickCount - the number of mouse clicks associated with event
popupTrigger - a boolean, true if this event is a trigger for a popup menu
button - which of the mouse buttons has changed state. NOBUTTON, BUTTON1, BUTTON2 or BUTTON3.

ii) public int getX()

- Returns the horizontal x position of the event relative to the source component.
- **Returns:** x an integer indicating horizontal position relative to the component

iii) public int getY()

- Returns the vertical y position of the event relative to the source component.
- **Returns:** y an integer indicating vertical position relative to the component

iv) public int getClickCount()

- Returns the number of mouse clicks associated with this event.
- **Returns:** integer value for the number of clicks

v) public int getButton()

- Returns which, if any, of the mouse buttons has changed state.
- **Returns:** one of the following constants: NOBUTTON, BUTTON1, BUTTON2 or BUTTON3.

r) MouseListener interface

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.

i) void mouseClicked(MouseEvent e)

- Invoked when the mouse button has been clicked (pressed and released) on a component.

ii) void mousePressed(MouseEvent e)

- Invoked when a mouse button has been pressed on a component.

iii) void mouseReleased(MouseEvent e)

- Invoked when a mouse button has been released on a component.

iv) void mouseEntered(MouseEvent e)

- Invoked when the mouse enters a component.

v) void mouseExited(MouseEvent e)

- Invoked when the mouse exits a component.

Example – 8 : abc.java

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class abc extends JFrame implements MouseListener
{
	JLabel lbl_mouse;
	abc()
	{
		super(" MouseEvent demo ");
		setBounds(100,100,500,500);
		setVisible(true);
		Container cp = getContentPane();
		cp.setLayout(null);
		lbl_mouse = new JLabel(" Play with mouse ");
		cp.add(lbl_mouse);
		lbl_mouse.setBounds(100,100,300,300);
		lbl_mouse.addMouseListener(this);
	}
	public void mouseClicked(MouseEvent e)
	{
		System.out.println(" Mouse Clicked ");
	}
	public void mousePressed(MouseEvent e)
	{
		System.out.println(" Mouse Pressed ");
	}
	public void mouseReleased(MouseEvent e)
	{
		System.out.println(" Mouse Released ");
	}
	public void mouseEntered(MouseEvent e)
	{
		System.out.println(" Mouse Enter ");
	}
```

```
	public void mouseExited(MouseEvent e)
	{
		System.out.println(" Mouse Exit ");
	}
	public static void main(String s[])
	{
		new abc();
	}
}
```

s) MouseMotionListener Interface

The listener interface for receiving mouse motion events on a component.

i) void mouseDragged(MouseEvent e)

- Invoked when a mouse button is pressed on a component and then dragged. MOUSE_DRAGGED events will continue to be delivered to the component where the drag originated until the mouse button is released

ii) void mouseMoved(MouseEvent e)

- Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Example – 9 : abc.java

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class abc extends JFrame implements MouseMotionListener
{
	JLabel lbl_mouse;
	abc()
	{
		super(" MouseEvent demo ");
		setBounds(100,100,500,500);
		setVisible(true);
		Container cp = getContentPane();
		cp.setLayout(null);
		lbl_mouse = new JLabel(" Play with mouse ");
		cp.add(lbl_mouse);
		lbl_mouse.setBounds(100,100,300,300);
		lbl_mouse.addMouseListener(this);
	}
```

```
    public void mouseMoved(MouseEvent e)
    {
        System.out.println(" Mouse Clicked ");
    }
    public void mouseDragged(MouseEvent e)
    {
        System.out.println(" Mouse Pressed ");
    }
    public static void main(String s[])
    {
        new abc();
    }
}
```

t) TextEvent class

A semantic event which indicates that an object's text changed. This high-level event is generated by an object (such as a TextComponent) when its text changes.

i) public TextEvent(Object source,int id)

- Constructs a TextEvent object.
- **Parameters: source** - the (TextComponent) object that originated the event
  id - an integer that identifies the event type

u) TextLIstener interface

The listener interface for receiving text events.

i) void textValueChanged(TextEvent e)

- Invoked when the value of the text has changed. The code written for this method performs the operations that need to occur when text changes.

v) WindowEvent class

A low-level event that indicates that a window has changed its status. This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transfered into or out of the Window.

There are ten types of window events. The WindowEvent class defines int constants that can be used to identify them. The constants and their meaning shown here:

- **WINDOW_ACTIVATED**

The window was activated.

- **WINDOW_CLOSED**

The window has been closed.

- **WINDOW_CLOSING**

The user requested that the window to be closed.

- **WINDOW_DEACTIVATED**

The window was deactivated.

- **WINDOW_DEICONIFIED**

The window was deiconified.

- **WINDOW_GAINED_FOCUS**

The window gained input focus.

- **WINDOW_ICONIFIED**

The window was iconified.

- **WINDOW_LOST_FOCUS**

The window lost input focus.

- **WINDOW_OPENED**

The window was opened.

- **WINDOW_STATE_CHANGED**

The state of the window changed.

i) public WindowEvent(Window source,int id)

- Constructs a WindowEvent object.
- **Parameters: source** - the Window object that originated the event
  id - an integer indicating the type of event

ii) publicWindowEvent(Window source,int id,Window opposite,int oldState, int newState)

- Constructs a WindowEvent object.
- Note that passing in an invalid id results in unspecified behavior. This method throws an IllegalArgumentException if source is null.
- **Parameters: source** - the Window object that originated the event
  id - an integer indicating the type of event.
  opposite - the other window involved in the focus or activation change, or null
  oldState - previous state of the window for window state change event
  newState - new state of the window for window state change event

iii) public Window getWindow()

- Returns the originator of the event.
- **Returns:** the Window object that originated the event

w) WindowListener interface

The listener interface for receiving window events.

i) void windowOpened(WindowEvent e)

- Invoked the first time a window is made visible.

ii) void windowClosing(WindowEvent e)

- Invoked when the user attempts to close the window from the window's system menu.

iii) void windowClosed(WindowEvent e)

- Invoked when a window has been closed as the result of calling dispose on the window

iv) void windowIconified(WindowEvent e)

- Invoked when a window is changed from a normal to a minimized state. For many platforms, a minimized window is displayed as the icon specified in the window's iconImage property.

v) void windowDeiconified(WindowEvent e)

- Invoked when a window is changed from a minimized to a normal state.

vi) void windowActivated(WindowEvent e)

- Invoked when the Window is set to be the active Window.

vii)void windowDeactivated(WindowEvent e)

- Invoked when a Window is no longer the active Window.

x) WindowFocusLIstener interface

The listener interface for receiving WindowEvents, including WINDOW_GAINED_FOCUS and WINDOW_LOST_FOCUS events.

i) void windowGainedFocus(WindowEvent e)

- Invoked when the Window is set to be the focused Window, which means that the Window, or one of its subcomponents, will receive keyboard events.

ii) void windowLostFocus(WindowEvent e)

- Invoked when the Window is no longer the focused Window, which means that keyboard events will no longer be delivered to the Window or any of its subcomponents.

2) Adapter classes

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

ComponentAdapter	:	ComponentListener
ContainerAdapter	:	ContainerListener
FocusAdapter	:	FocusListener
KeyAdapter	:	KeyListener
MouseAdapter	:	MouseListener
MouseMotionAdapter	:	MouseMotionListener
WindowAdapter	:	WindowListener

Example – 1 : abc.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class A extends JFrame
{
	A(String tit)
	{
		super(tit);
		Container cp = getContentPane();
		cp.setLayout(null);
		setBounds(100,100,500,500);
		setVisible(true);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		addMouseListener(new B(this));
	}
}
class B extends MouseAdapter
{
	A a1;
	static int i=0;
	B(A a)
	{
		a1=a;
	}
```

```
	public void mouseClicked(MouseEvent me)
	{
		System.out.println(" Hello fp : "+ i++);
	}
}
class abc
{
	public static void main(String s[])
	{
		new A(" MouseAdapter demo ");
	}
}
```

Example – 2 : abc.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class abc extends JFrame
{
	static int i=0;
	abc(String s)
	{
		super(s);
		Container cp = getContentPane();
		cp.setLayout(null);
		setBounds(100,100,500,500);
		setVisible(true);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		addMouseListener(new MouseAdapter()
		{
			public void mouseClicked(MouseEvent me)
			{
				System.out.println(" Hello fp : "+ i++);
			}
		});
	}
	public static void main(String s[])
	{
		new abc(" MouseAdapter demo ");
	}
}
```