GHARDA FOUNDATION'S
GHARDA INSTITUTE OF TECHNOLOGY
A/P:-LAVEL, TALUKA: KHED, DIST. RATNAGIRI, STATE: MAHARASTRA

LABORATORY MANUAL

SUBJECT: - Object Oriented Programming Methodology

Class: - S.E. (COMP)

Semester: - III

Prepared by
Prof. V.M.Nair
Asst. Professor
Computer Engineering Dept

List of Experiments	
Sr.No	**Title**
1	Program on various ways to accept data through keyboard using Scanner class
2	Program to create class with members and methods, accept and display details for single object.
3	Program on constructor overloading
4	Program on method overloading
5	Program on One Dimensional array
6	Study of Two Dimensional arrays.
7	Study of String class & its methods.
8	Study of StringBuffer class & its methods.
9	Study of Inheritance and its various types.
10	Study of Interface and to illustrate the multiple inheritance by using Interfaces.
11	Study of Packages with a program to demonstrate use of user defined packages.
12	Program on to demonstrate use of Exception handling by divide of zero situations.
13	Program to demonstrate Multithreading by implementing thread using runnable interface.
14	Study of Applet with a program to display various shapes in some different colors.
15	Java program for Bouncing of a Ball using applet.
16	Study of Abstract Window toolkit with a Program to display simple Textfields & Button controls on Frame. Display First Name/Last name on textField according to event called(Botton Clicked by User.)
17	Write AWT Program to create simple arithmetic calculator by using Event Handling.

Experiment No. 1

Aim: - Program on various ways to accept data through keyboard using Scanner class

Theory :-

We may also give values to the variables through by using various classes and their methods. One of the useful classes among them is **Scanner.** It is added by jdk1.5.

This class is defined in one of the library package of Java i.e. java.util. So in order to use is in our program we have to use following import statement at the top of our program.

import java.util.Scanner;

After this, the class Scanner will be available for our use in the program. We have to create the object of scanner in following way.

Scanner in = new Scanner(System.in);

System.in is standard input handle i.e. keyboard. 'in' is the object of the Scanner class, which is then used to read any type of the data from keyboard in it. We have to use any of the following methods according to the requirement.

nextBoolean() for reading boolean values

nextByte() for reading byte values

nextFloat() for reading float values

nextDouble() for reading double values

nextInt() for reading int values

nextShort() for reading short values

nextLong() for reading long values

Program :-

```
import java.util.Scanner;
class ScannerTest
{

	public static void main(String args[])
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter your rollno");
		int rollno=sc.nextInt();
		System.out.println("Enter your name");
		String name=sc.next();
		System.out.println("Enter your fee");
		double fee=sc.nextDouble();
		System.out.println("\nRollno:"+rollno+"\nName:"+name+"\nFee:"+fee);
	}

}
```

Output :-

```
D:\VJ \jvm>javac ScannerTest.java
D:\VJ \jvm> java ScannerTest
Enter your rollno
 111
 Enter your name
 Ratan
 Enter your fee
 95000

 Rollno:111
 Name:Ratan
 Fee:950000
```

Experiment No. 2

Aim: - Program to create class with members and methods, accept and display details for single object.

Theory:-

Class definition

A class is a user-defined data type with the template that serves to define its properties. Most important thing here is to understand is that a class defines new data type. Once defined, this new data type can be used to create objects of that type. Thus, class is a template for an object and object is the instance of the class. Object can be called as variable of type class. When we define a class, we declare its exact form and nature.

Here we specify the data that it contains and the code that operates on data. Very simple classes may contain only the code or only data, most real world classes contain both.

A class is declared by the keyword **class**. Class definition form is given below.

```
class class-name
{
data-type instance-variable1;
data-type instance-variable2;
- - - - -;
data-type instance-variableN;

data-type method-name1(parameter-list)
{
//body of the method1
}
data-type method-name2(parameter-list)
{
//body of the method2
}
- - - - -;
data-type method-nameN(parameter-list)
{
//body of the methodN
```

```
}
}
```

Adding variables to a class

The data in a class is in the form of instance variables. We can declare the instance variables exactly the same way as we declare the local variables. For example:

```
class Country
{
long population;
float currencyRate;
int noOfStates;
}
```

Creating objects

Creating the objects of the class is also called as instantiating an object. As stated earlier, a class creates new data type. In the above case of example 'Country' is new data type.

Objects are created using the 'new' operator. This 'new' operator creates an object of the specified class and returns the reference of that object. See the example for creating the object of the above class.

```
Country japan; // declaration
japan = new Country(); // instantiation
```

The first statement declares a variable to hold the object reference. Second statement actually assigns the object reference to the variable.

The variable name 'japan' is called as the object of class 'Country'. Both statements can also be combined into one as shown below:

```
Country japan = new Country();
```

Here, Country ( ) specifies the default constructor of the class. We can create any number of objects of the class 'Country' as,

```
Country nepal = new Country();
Country bhutan = new Country();
Country myanmar = new Country();
```

After the objects are created, there is a separate copy of the instance variables is created for each of them. So, separate memory is allocated to them.

Program:-

```
import java.io.*;
import java.util.*;
class Circle
{
        double p;
        double a;
        void area(double r)
        {
        a=3.14*r*r;
        System.out.println("\n Area of rectangle= "+a);
        }
        void perimeter(double r)
        {
        p=2*3.14*r;
        System.out.println("\n Perimeter of circle= "+p);
        }
}
class Myprogram
{
        public static void main(String args[])
        {
                double r;
                Scanner in = new Scanner(System.in);
                Circle m=new Circle();
                System.out.println("Enter radius");
                r= in.nextDouble();
                m.area(r);
                m.perimeter(r);
    }
}
```

Output:-

```
D:\VJ \jvm>javac Myprogram.java
D:\VJ \jvm>java Myprogram
Enter radius
5
Area of Rectangle=78.5
Perimeter of Circle=31.4
```

Experiment No. 3

Aim: - Write a program to demonstrate Constructor Overloading.

Theory: -

Constructor

It can be boring to initialize all of the variables in a class each time an instance is created. Even then we add convenience functions like input( ), it would be simpler and more concise to have all of the setup done at the time the object is first created. Because the requirement for initialization is so common, Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.

A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and it is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the **new** operator completes. Constructors look a little strange because they have no return type, not even **void**. This is because the implicit return type of a class' constructor is the class type itself. It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object instantly.

Constructor overloading

The concept of overloading refers to the use of same name for different purposes. In simple language, a class can have more than one constructor but with different parameters. This is known as constructor overloading.

Program:-

```
import java.io.*;
import java.util.Scanner;
class Box
{
        int height;
        int  depth;
        int length;
        Box()
        {
        height = depth = length = 10;
        }
        Box(int x,int y)
        {
        height = x;
```

```
        depth = y;
        }
        Box(int x, int y, int z)
        {
        height = x;
        depth = y;
        length = z;
        }
}
class ConstrOverloading
{
        public static void main(String args[])
        {
                Box a = new Box(); //statement1
                System.out.println("depth of a : "+a.depth);
                Box b = new Box(12,23); //statement2
                System.out.println("depth of b : "+b.depth);
                Box c = new Box(99,84,36); //statement3
                System.out.println("depth of c : "+c.depth);
        }
}
```

Output:-

```
D:\Modani D.G.\new programs>javac ConstrOverloading.java
D:\Modani D.G.\new programs>java ConstrOverloading
depth of a : 10
depth of b : 23
depth of c : 84
```

Experiment No. 4

Aim: - Write a program using method overloading to find and display area of circle and rectangle

Theory:-

Methods overloading

In Java it is possible to define two or more methods within the same class that are having the same name, but their parameter declarations are different. In the case, the methods are said to be overloaded, and the process is referred to as method overloading. Method overloading is one of the ways of Java implementation of polymorphism. This concept of method overloading is just same as function overloading of C++. If we have never used a language that allows the overloading of methods, then the concept may seem strange at first. But, method overloading is one of Java's most exciting and useful features When an overloaded method is called, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call. Thus, overloaded methods must differ in the type and/or number of their parameters. While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method. When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call

Program:-

```
import java.io.*;
import java.util.*;
class Meth
{
	int rarea;
	double carea;
	void area(int l, int b)
	{
		rarea=l*b;
		System.out.println("\n Area of rectangle= "+rarea);
	}
	void area(double r)
	{
		carea=3.14*r*r;
		System.out.println("\n Area of circle= "+carea);
	}
}
class Methdover
{
	public static void main(String args[])
	{
```

```
                int l,b;
                double r;
                Scanner in = new Scanner(System.in);
                Meth m=new Meth();
                System.out.println("Enter length & Breath");
                l= in.nextInt();
                b=in.nextInt();
                m.area(l,b);
                System.out.println("Enter radius");
                r= in.nextDouble();
                m.area(r);
        }
}
```

```
Output:-
D:\VJ \jvm>javac Methdover.java D:\VJ \jvm>java
Methdover
Enetr lenghth & Breath
10
20
Area of rectangle= 200
Enter radius
2.3
Area of circle= 16.610599999999998
```

Experiment No. 5

Aim: - Study of one Dimensional array.

Theory:-

Arrays

An array is contiguous or related data items that share the common name. It offers the convenient means of grouping information. This means that all elements in the array have the same data type. A position in the array is indicated by a non-negative integer value called as index.

An element at the given position is accessed by using this index. The size of array is fixed and cannot increase to accommodate more elements

One dimensional arrays

One dimensional array is essentially, a list of same-typed variables. A list of items can be given one variable name using only one subscript. The general form of creating one dimensional array is,

datatype variablename[] = new datatype[size];

example,

int val[] = new int[5];

This declaration will create five different variables in a contiguous memory locations referred by the common name 'val'. The first element among them is val[0] to val[4]. Remember the index number of the array always start with 0.

Array initialization

When an array is created, each element of the array is set to the default initial value for its type i.e. zero for all numeric types, '\u0000' for char type, false for Boolean and null for reference types. Putting the values inside the array is called as array initialization.

At compile time the individual element can be initialized as,

arrayname[index] = value;

or

arrayname[index] = expression;

For example:

```
val[0] = 36;
val[1] = 26;
val[2] = 78;
val[3] = 3;
val[4] = 95;
val[1] = (a + b) * 2;
val[0] = 95 / 19;
```

Array length

All arrays store the allocated size in an attribute named length. We can treat the array as object in order to access number of elements of it. Such as,

For example:

```
int arr[] = {8, 6, 2, 4, 9, 3, 1};
int size = arr.length;
```

After the creation of the array 'arr', compiler will automatically decide the size of the array. This size or length or number of elements of the array can be obtained using attribute 'length' as shown above. The variable 'size' will contain value 7.

Program:- WAP to sort the given array in Ascending & Descending order.

```
import java.io.*;
import java.util.*;
class Asc
	{
	public static void main(String Args[])
		{
		Scanner in=new Scanner(System.in);
		int a[]=new int[5];
		int i,j,temp;
		System.out.println("Enter Any 5 Elements for Array.");
		for(i=0;i<a.length;i++)
			{
			a[i]=in.nextInt();
			}
		for(i=0;i<a.length-1;i++)
			{
			for(j=0;j<(a.length-1);j++)
				{
				if(a[j+1]<a[j])
					{
					temp=a[j];
					a[j]=a[j+1];
					a[j+1]=temp;
					}
				}
			}
		System.out.println("The Ascending order is");
		for(i=0;i<a.length;i++)
			{
			System.out.println(a[i]);
			}
		for(i=0;i<(a.length-1);i++)
			{
			for(j=0;j<(a.length-1);j++)
				{
				if(a[j+1]>a[j])
					{
					temp=a[j];
					a[j]=a[j+1];
					a[j+1]=temp;
					}
				}
			}
		System.out.println("The Descending Order is");
		for(i=0;i<a.length;i++)
			{
			System.out.println(a[i]);
```

```
                }
            }
        }
```

```
D:\ VJ \jvm>javac Asc.java
D:\ VJ \jvm>java Asc
Enter Any 5 Elements for Array.
10
50
20
40
30
The Ascending order is
10
20
30
40
50
The Descending Order is
50
40
30
20
10
```

Experiment No. 6

Aim: - Study of Two Dimensional arrays.

Theory:-

Two dimensional arrays

It is also called as array of arrays. Many times it is required to manipulate the data in table format or in matrix format which contains rows and columns. In these cases, we have to give two dimensions to the array. That is, a table of 5 rows and 4 columns can be referred as, table[5][4] The first dimension 5 is number of rows and second dimension 4 isnumber of columns. In order to create such two dimensional arrays in Java following syntax should be followed.

datatype arrayname[][] = new datatype[row][column];

For example:

int table[][] = new int[5][4];

Program: - WAP to sort the multiply two matrices.

```
import java.io.*;
import java.util.*;
class Matmul
        {
        public static void main(String Args[])
                {
                int m,i,j,k;
                Scanner in=new Scanner(System.in);
                System.out.println("Enter no of Row/ Column of Square Matrix.");
                m=in.nextInt();
                int first[][]=new int[m][m];
                int second[][]=new int[m][m];
                int third[][]=new int[m][m];
                System.out.println("Enter "+(m*m)+" Elements of first Matrix");
                for(i=0;i<m;i++)
                        {
                        for(j=0;j<m;j++)
                                {
                                first[i][j]=in.nextInt();
                                }
                        }
                System.out.println("Enter "+(m*m)+" elements of Second matrix");
                for(i=0;i<m;i++)
                        {
                        for(j=0;j<m;j++)
                                {
                                second[i][j]=in.nextInt();
                                }
                        }
                System.out.println("The matrix Multiplication is");
                for(i=0;i<m;i++)
                        {
```

```
                for(j=0;j<m;j++)
                        third[i][j]=0;
                        for(k=0;k<m;k++)
                                {
                                third[i][j]=third[i][j]+first[i][k]*second[k][j];
                                }
                        }
                }
        for(i=0;i<m;i++)
                {
                for(j=0;j<m;j++)
                        {
                        System.out.print(third[i][j]+"\t");
                        }
                System.out.print("\n");
                }
        }
}
```

Output:-

```
D:\ VJ \jvm>javac Matmul.java

D:\ VJ \jvm>java Matmul
Enter Row/Column of Square Matrix.
3
Enter 9 Elements of first Matrix
1 2 3
4 5 6
7 8 9
Enter 9 elements of Second matrix
1 1 1
1 1 1
1 1 1
The matrix Multiplication is
6   6   6
15  15   15
24  24   24
```

Experiment No. 7

Aim: - Study of String class & its methods.

Theory:-

String class

As is the case in most other programming languages, in Java a string is a sequence of characters. but, unlike many other languages that implement strings as character arrays, Java implements strings as objects of type String. Implementing strings as built-in objects allows Java to provide a full complement of features that make string handling convenient. For example, Java has methods to compare two strings, search for a substring, concatenate two strings, and change the case of letters within a string. Also, String objects can be constructed a number of ways, making it easy to obtain a string when needed. Somewhat unexpectedly, when we create a String object, we are creating a string that cannot be changed. That is, once a String object has been created, you cannot change the characters that comprise that string. At first, this may seem to be a serious restriction. However, this is not the case. We can still perform all types of string operations. The difference is that each time we need an altered version of an existing string; a new String object is created that contains the modifications. The original string is left unchanged.

Creating the strings

String s = new String();
For example:
char chars[] = {'h', 'e', 'l', 'l', 'o'};
String s = new String(chars);
Direct initialization is also possible as,
String s = "Hello"

String operations and methods

Following are the methods of the String class.

length( )

This method is used to find total number of characters from the string.
Syntax:
int length()
Example:
String s = "Pramod"; int
len = s.length();
This will print the value 6 which is the length of string 's'.

concat( )

This is used to join two strings.
Syntax:
String concat(String str)
Example:
String s = "First"; String
t = "Second";
s.concat(t);
The contents of the string 's' will be "FirstSecond" after the execution of these statements. The operator + can also be used to join two strings together. We have used this operator several

times in program for the same purpose. For example:
String s = "How " + "are " + "you ?";
String x = "Number = " + 2;

charAt
This method extracts a single character from the string.
Syntax:
char charAt(int index)
It extracts the character of invoking String from position specified by 'index'.
Example:
String s = "Indian";
char ch = s.charAt(2); //ch will be 'd' after this

getChars( )
It extracts the sequence of characters from the string.
Syntax:
void getChars(int s, int e, char target[], int tstart)
This will extract the characters from position s to e from invoking string and store it in character array target from position 'tstart'.
Example:
String s = "I Love Java";
char ch[] = new char[4];
s.getChars(2, 5, ch, 0); //ch will contain "love"

startsWith( ) and endsWith( )
The method 'startsWith( )' determines whether a given string starts with specified string or not. Conversely, the 'endsWith( )' determines whether a given string ends with specified string or not.
Syntax:
boolean startsWith(String str)
boolean endsWith(String str)
boolean startsWith(String str, int startIndex)
Here, str is the string to be checked and startIndex is the index from which the comparison is to be made.
Example:
boolean a = "camera".startsWith("cam"); //true
boolean b = "camera".endsWith("mere"); //false

compareTo( ) and compareToIgnoreCase( )
For sorting applications, we need to know which string is less than, equal to, or greater than the next. In such cases the compareTo( ) or compareToIgnoreCase() method can be used. It is used to check whether one string is greater than, less than or equal to another string or not.
First compareTo( ) is case-sensitive and second form compareToIgnoreCase( ) is case insensitive.
Syntax:
int compareTo(String str) int compareToIgnoreCase(String str) Here,
str is the string to be compared with invoking object's string. It will
return an integer value. When,
value<0: invoking string is less that str value>0: invoking string is greater that str value=0: both the strings are equal

Example:

```
String str1 = "catch";
String str2 = "match";
if(str1.compareTo(str2)<0)        //true
System.out.println("str2  is  greater");
else
System.out.println("str1 is greater");
```

indexOf( )

If we want to search a particular character or substring in the string then this method can be used. This method returns the index of the particular character or sting which is searched. It can be used in different ways.

int indexOf(int ch)

This method searches first occurrence of the character from the string.

int indexOf(String str)

This method searches first occurrence of the substring from another string.

int indexOf(int ch, int startIndex)

int indexOf(String str, int startIndex)

Here the 'startIndex' specifies the index at which point of search begins.

Example:

```
String  s  =  "Maharashtra";
String t = "Tamilnadu";
int in = s.indexOf('a'); //in will be 1
int  mn  =  t.indexOf('a',  2);  //mn  will  be  6
int x = s.indexOf("tra"); //x will be 8
```

lastIndexOf( )

This method searches the last occurrence of the particular character or string in the invoking object's string and returns index of that. Like indexOf( ) this method is also having four different forms.

int lastIndexOf(int ch)

int lastIndexOf(String str)

int lastIndexOf(int ch, int startIndex)

int lastIndexOf(String str, int startIndex)

Here 'ch' is the character and 'str' is the string to be searched inside the invoking string from last. In case of last two forms of the method, the search begins from 'startIndex' to zero.

```
String  s  =  "Maharashtra";
String t = "Tamilnadu";
int in = s.lastIndexOf('a'); //in will be 10
int  mn  =  t.lastIndexOf('a',  2);  //mn  will  be  1
int x = s.lastIndexOf("ra"); //x will be 9
```

substring( )

It is used to extract a substring from the string. It has two different forms:

String substring(int start)

String substring(int start, int end)

The first form extracts substring from 'start' to end of the string. Second form extracts the substring from position 'start' to the position 'end' of the invoking string.

Example:

String str = "Are you a Marathi guy";
String s = s.substring(10); //s will be "Marathi guy"
String t = s.substring(10,16); //t will be "Marathi"

replace( )

This method is used to replace all the occurrences of a character with another character of the string. It has following form:
Syntax:
String replace(char original, char replacement)
Here 'original' is character to be replaced and 'replacement' is new character exchanged instead of that.
Example:
String s = "mama".replace('m','k');
It will initialize the string 's' with value "kaka".

trim( )

It is used to remove the leading and trailing white spaces from the string. Remember, it does not remove the spaces in between the characters of the string.
Syntax:
String trim()
Example:
String s = " What is this? ";
String k = s.trim();
After this, the string 'k' will contain "What is this?"

toLowerCase( ) and toUpperCase( )

These methods will convert the strings into lower case and upper case respectively. Non-alphabetical characters such as digits and symbols are unaffected.
Syntax:
String toLowerCase( )
String toUpperCase( )
Example:
String s = "Pramod and Kunda";
String t = s.toUpperCase(); String
u = s.toLowerCase();
After execution of these statements, the value of t will be "PRAMOD AND KUNDA" and u will be "pramod and kunda".

Program1:- Write a program to input a string and count number of vowels in the string.

```
import java.util.*;
class vowels
{
	public static void main(String args[])
	{
		int i,count=0,consonant;
		Scanner sc=new Scanner(System.in);
		String s1;
		System.out.println("Enter your string");
		s1=sc.nextLine();
		int len=s1.length();
		for(i=0;i<len;i++)
		{
			if(s1.charAt(i)=='a'  ||  s1.charAt(i)=='e'  ||  s1.charAt(i)=='i'  ||
			s1.charAt(i)=='o'  ||  s1.charAt(i)=='u'  ||  s1.charAt(i)=='A'  ||
			s1.charAt(i)=='E'  ||  s1.charAt(i)=='I'  ||  s1.charAt(i)=='O'  ||
			s1.charAt(i)=='U' )
			{
				count++;
			}
		}
		System.out.println("Toal number of vowels present in the given string= "
		+count);
		consonant=len-count;
		System.out.println("Toal number of consonant present in the given string= "
		+consonant);
	}
}
```

```
D:\VJ \jvm>javac vowels.java
D:\ VJ \jvm>java vowels
Enter a string
Sachin
Toal number of vowels present in the given string= 2
Toal number of consonant present in the given string=4
```

Program No 2. Write a program to input a string and check if it is a palindrome.

Definition of Palindrome:- If we read the given word in reverse manner & it looks like original word then such words are called as Palindrome.
Eg. Madam, malayalam,nitin etc.

```
import java.util.*;
class Palindrome
{
        public static void main(String args[])
        {
                String original, reverse="";
                Scanner in = new Scanner(System.in);
                System.out.println("Enter a string to check if it is a palindrome");
                original = in.nextLine();
                int length = original.length();
                for ( int i = length - 1 ; i >= 0 ; i-- )
                        reverse = reverse + original.charAt(i);
                if (original.equals(reverse))
                      System.out.println("Entered string is a palindrome.");
                else
                      System.out.println("Entered string is not a palindrome.");
         }
 }
```

```
Output:-
D:\VJ \jvm>javac Palindrome.java
D:\VJ \jvm>java Palindrome
Enter a string
NITIN
Entered string is a palindrome
```

Experiment No. 8

Aim: - Study of StringBuffer class & its methods.

Theory:-

StringBuffer class

String represents fixed-length, immutable character sequences. Incontrast, StringBuffer represents growable and writeable character sequences. It is possible to insert the characters anywhere inside theStringBuffer. StringBuffer will automatically grow to make room for such additions and often has more characters pre-allocated than are actually needed, to allow room for growth.

Creating StringBuffer

StringBuffer class defines three different constructors:
StringBuffer()
StringBuffer(int size)
StringBuffer(String str)
The first form i.e. default constructor reserves room for 16characters without reallocation. The second form accepts an integerargument that explicitly sets the size of the buffer with given value. The third form accepts a String argument that sets the initial contents of the
StringBuffer object and reserves room for 16 more characters without reallocation. StringBuffer allocates room for 16 additional characters when no specific buffer length is requested, because reallocation is a costly process in terms of time. Also, frequent reallocations can fragment memory. By allocating room for a few extra characters, StringBuffer reduces the number of reallocations that take place.

StringBuffer operations and methods

Several methods of classes String and StringBuffer are same but as per the functionality, StringBuffer has been added with some special methods.

length( ) and capacity( )

The 'length( )' finds total number of characters that a StringBuffer is having in it. The total allocated capacity (no. of characters) can be found using the capacity( ) method.
Syntax:
int length() int
capacity()
Example:

```
StringBuffer sb = new StringBuffer("Oriya");
int len = sb.length()); //len will be 5
int cap = sb.capacity()); //cap will be 21
```

Here the variable len will contain total number of characters i.e. 5 and cap will contain capacity 21 i.e. actual length + additional room for 16 characters.

ensureCapacity( )

If we want to pre-allocate room for a certain number of characters after a StringBuffer has been created, we can use ensureCapacity( ) to set the size of the buffer. This is useful if we know in advance that we will be appending a large number of small strings to a StringBuffer. ensureCapacity( ) has this general form:
void ensureCapacity(int capacity)

Here, capacity specifies the size of the buffer.
Example:
StringBuffer sb = new StringBuffer(10);
sb.ensureCapacity(20);

setLength( )
It is used to set the length of the buffer within a StringBuffer object. Its general form is:
void setLength(int len)
Here, len specifies the length of the buffer. This value must be nonnegative. When we increase the size of the buffer, null characters are added to the end of the existing buffer. If we call setLength( ) with a value less than the current value returned by length( ), then the characters stored beyond the new length will be lost.

append( )
The append( ) method concatenates the string representation of any other type of data to the end of the invoking StringBuffer object. It has overloaded versions for all the built-in types and for Object. Following are a few of its forms:
StringBuffer append(String str)
StringBuffer append(int num)
StringBuffer append(Object obj)
Example:
StringBuffer sb = new StringBuffer("cricket");
int x = 52;
float f = 13.4f;
StringBuffer a = sb.append(x); //a will be cricket52
StringBuffer b = sb.append(f); //b will be cricket5213.4

insert( )
This method is used to insert one string, character or object into another string. Forms of this method are:
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)
Here the 'index' specifies the index position at which point the string will be inserted into the invoking StringBuffer object.
Example:
StringBuffer sb = new StringBuffer("I Java!");
sb.insert(2, "love ");
After this, the contents of 'sb' will be "I love Java!"

reverse( )
We can reverse the characters inside the StringBuffer using reverse( ) method.
Syntax:
StringBuffer reverse()
This method returns the reversed object upon which it is called.
Example:
StringBuffer s = new StringBuffer("Yellow");
s.reverse(); //s will be "wolleY"

delete( ) and deleteCharAt( )

These methods are used to delete a single character or a sequence of characters from the StringBuffer.
Syntax:
StringBuffer delete(int startIndex, int endIndex)
StringBuffer deleteCharAt(int loc)

The delete( ) method deletes a sequence of characters from the invoking object. Here, 'startIndex' specifies the index of the first character to remove, and 'endIndex' specifies an index one past the last character to remove. Thus, the substring deleted runs from 'startIndex' to 'endIndex'–1. The resulting StringBuffer object is returned. The deleteCharAt( ) method deletes the character at the index specified by 'loc'. It returns the resulting StringBuffer object.

Example:
StringBuffer sb = new StringBuffer("Chandramukhi");
sb.delete(4, 7); //sb will be "Chanmukhi"
sb.deleteCharAt(0); //sb will be "hanmukhi"

The methods such as getChars, replace, substring, indexOf and lastIndexOf are having the same syntax and use as that of String class except here the manipulation is with StringBuffer only.

Program:- WAP to demonstrate methods of StringBuffer class.

```
import java.io.*;
import java.util.*;

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

        Scanner in = new Scanner(System.in);
        StringBuffer str = new StringBuffer("India");
        System.out.println("\n1.Length\n2.Capacity\n3.Setlength\n4.Charat\n5.Setcharat\n6
        .Append\n7.Deletecharat\n8.Substring1\n9.Substring2\n10.Insert\n11.Reverse");
        System.out.println("Enter ur chioce");
        int ch=in.nextInt();
        switch(ch)
        {
                case 1:
                        System.out.println("The length of the string is:" +str.length());
                        break;
                case 2:
                        System.out.println("The capacity of String:" + str.capacity());
                        break;
                case 3:
                        str.setLength(15);
                        System.out.println("Set length of String:" + str.length());
                        break;
```

```
                case 4:
                        System.out.println("The character at 2nd position:" + str.charAt(2));
                        break;
                case 5:
                        str.setCharAt(2,'u');
                        System.out.println("The string after setting position is:" +str);
                        break;
                case 6:
                        System.out.println("The string after appending:" + str.append("
                        Sohni"));
                        break;
                case 7:
                        System.out.println("The string after deletion:" + str.deleteCharAt(1));
                        break;
                case 8:
                        System.out.println("The substring1 is:" + str.substring(2));
                        break;
                case 9:
                        System.out.println("The subsstring2 is:" +str.substring(2,4));
                        break;
                case 10:
                        System.out.println(" The string after insertion is:" + str.insert(1,'m'));
                        break;
                case 11:
                        System.out.println("The string after reverse is:" + str.reverse());
                        break;
                default:
                        System.out.println("You have Entered wrong choice");
                        break;
        }
    }
}
```

Output:-

```
D:\VJ \jvm>javac s.java
D:\VJ \jvm>java s
1.Length
2.Capacity
3.Setlength
4.Charat
5.Setcharat
6.Append
7.Deletecharat
8.Substring
9.Substring1
10.Insert
11.Reverse
Enter ur chioce
1
The length of the string is:5
Enter ur chioce
2
The capacity of String:21
Enter ur chioce
3
Set length of String:15
Enter ur chioce
4
The character at 2nd position:d
Enter ur chioce
5
The string after setting position is:Inuia
Enter ur chioce
6
The string after appending:India Sohni
Enter ur chioce
7
The string after deletion:Idia
Enter ur chioce
8
The substring is:dia
Enter ur chioce
9
The subsstring2 is:di
Enter ur chioce
10
The string after insertion is:Imndia
Enter ur chioce
11
The string after reverse is:aidnI
```

Experiment No. 9

Aim: - Study of Inheritance and it's various types.

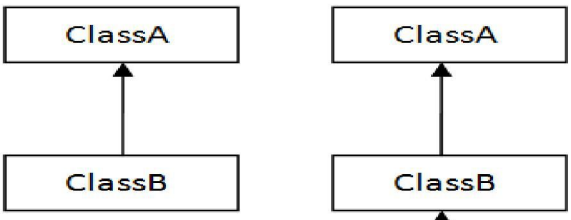
Theory:-

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called **inheritance**. The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.

Types of inheritance in java

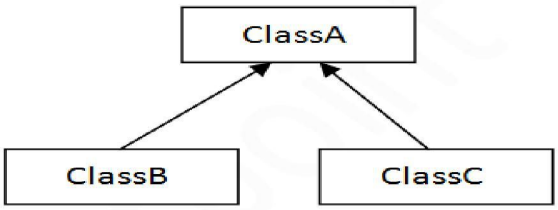
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

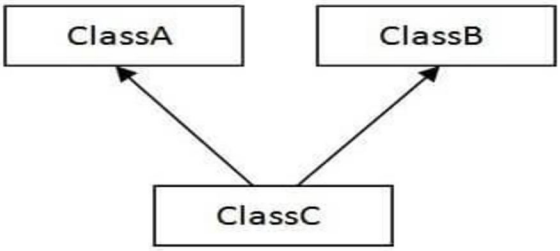


1) Single

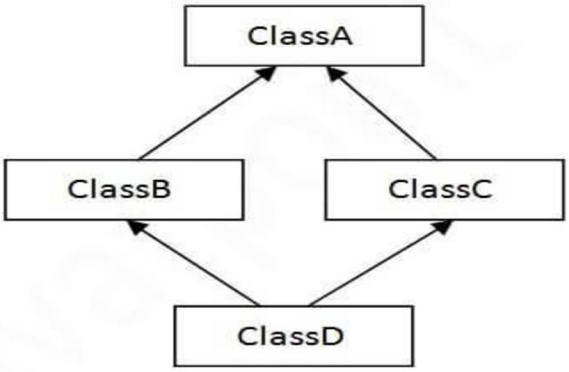
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

1) Single Inheritance

Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.

2) Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.

3) Hierarchical Inheritance

In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D.

4) Multiple Inheritance

"**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.

Note 1: Multiple Inheritance is very rarely used in software projects. Using Multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

Note 2: Most of the new OO languages like **Small Talk, Java, C# do not support Multiple inheritance**. Multiple Inheritance is supported in C++.

5) Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple** inheritance. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! By using **interfaces** we can have multiple as well as **hybrid inheritance** in Java.

Program: - WAP to implement Single Inheritance.

```
class Rectangle
        {
           int l,b;
          void Setval(int x,int y)
           {
              l=x;
              b=y;
           }
               int GetRect()
                {
                   return l*b;
                }
        }
                class Triangle extends Rectangle
                   {
                      int b,h;
                      float a;
                      void SetData(int v,int u)
                         {
                            b=u;
                            h=v;
                         }
                            float GetTri()
                              {
                                 a=(float)l/2*b*h;
                                 return (a);
                              }
                    }
                           class SingleInheritance
                             {
                                 public static void main(String args[])
                                    {
                                       Triangle Tri=new Triangle();
                                       Tri.Setval(50,8);
                                       Tri.SetData(17,7);
                                       System.out.println("Area of Rectangle is :" +Tri.GetRect());
                                       System.out.println("Area of Triangle is :"+Tri.GetTri());
                                    }
                             }
```

```
Output:-
D:\VJ \jvm>javac SingleInheritance.java
D:\VJ \jvm>java SingleInheritance
Area of Rectangle is :400
Area of Triangle is :2975.0
```

Program: - WAP to implement Multilevel Inheritance.

```
class WorkerDetail
        {
            int c,s;
            String n;
            float h;
            void setSalary(int x, String y, int z)
             {
                 c=x;
                 n=y;
                 s=z;
             }
                void showDetail()
                 {
                     System.out.println("Code :"+ c);
                     System.out.println("Name : "+n);
                     System.out.println("Salary "+s);
                 }
                     void getHra()
                      {
                          h=(float)s*60/100;
                          System.out.println("HRA :"+h);
                      }
        }
             class OfficerSal extends WorkerDetail
              {
                float d;
                void getDA()
                  {
                      d=(float)s*98/100;
                      System.out.println("DA :"+d);
                  }
              };
                     class ManagerSal extends OfficerSal
                      {
                     float ca,g;
                     void getCA()
                         {
                                ca=(float)s*20/100;
                            System.out.println("City Allowance :"+ca);
                                }
                                       void getgross()
                                        {
                                           g=s+h+d+ca;
                                           System.out.println("Gross Salary :"+g);
                                        }
                           }
```

```
class Multilevel
   {
        public static void main(String args[])
  {
    ManagerSal m=new ManagerSal();
    m.setSalary(11, "Ankit Rana",13000);
    System.out.println("Details of Manager is :");
    m.showDetail();
    m.getHra();
    m.getDA();
    m.getCA();
    m.getgross();
  }
   }
```

```
Output:-
D:\VJ \jvm>javac Multilevel.java
D:\VJ \jvm>java Multilevel
Details of Manager is
Code :11
Name : Ankit Rana
Salary 13000
HRA :7800.0
DA :12740.0
City Allowance :2600.0
Gross Salary :36140.0
```

Program: - WAP to implement Hierarchical Inheritance.

```
class ClassA
{
   public void dispA()
   {
      System.out.println("disp() method of ClassA");
   }
}
class ClassB extends ClassA
{
   public void dispB()
   {
      System.out.println("disp() method of ClassB");
   }
}
class ClassC extends ClassA
{
   public void dispC()
   {
```

```
        System.out.println("disp() method of ClassC");
    }
}
class ClassD extends ClassA
{
    public void dispD()
    {
        System.out.println("disp() method of ClassD");
    }
}
class HierarchicalInheritanceTest
{
    public static void main(String args[])
    {
        //Assigning ClassB object to ClassB reference
        ClassB b = new ClassB();
        //call dispB() method of ClassB
        b.dispB();
        //call dispA() method of ClassA
        b.dispA();
        //Assigning ClassC object to ClassC reference
        ClassC c = new ClassC();
        //call dispC() method of ClassC
        c.dispC();
        //call dispA() method of ClassA
        c.dispA();

        //Assigning ClassD object to ClassD reference
        ClassD d = new ClassD();
        //call dispD() method of ClassD
        d.dispD();
        //call dispA() method of ClassA
        d.dispA();
    }
}
```

```
Output:-
D:\VJ \jvm>javac HierarchicalInheritanceTest.java
D:\VJ \jvm>java HierarchicalInheritanceTest
disp() method of ClassB
disp() method of ClassA
disp() method of ClassC
disp() method of ClassA
disp() method of ClassD
disp() method of ClassA
```

Experiment No. 10

Aim: - Study of Interface and to illustrate the multiple inheritance by using Interfaces.

Theory:-

An interface is a **blueprint of class**. It has constants and abstract methods. We will see in the past abstraction tutorial interface give **100% abstraction**, so it is called fully abstraction.

The interface definition states the names of the methods and their return types and argument signatures. There is no executable body for any method that is left to each class that implements the interface.

Interface also represent **IS-A** relationship.

Why use interface?
- It is use to achieve **fully abstraction**.
- By using interface we achieve **multiple inheritances**.

Syntax:
The interface keyword is used to declare an interface.

```
public interface Interface-name{

        //Any number of final, static fields
        //Any number of abstract method
declarations
}
```

Notes on Interfaces:

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor (as it cannot be used to create objects)

Why And When To Use Interfaces?

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces.
Note: To implement multiple interfaces, separate them with a comma (see example below).

```
* Example on the multiple inheritance with interface. */

class A {
    protected int i = 1000;
    public void print() { System.out.println("I am from A "+i);
    }
}

interface C {
    public static int j = 555;
    void printInterfaceC();
}

interface D {
    public static int k = 666;
    void printInterfaceD();
}

class B extends A implements C, D{
     public int aValue = 999;
     public void printInterfaceC() {
          System.out.println("I am from C " + j);
     }
     public void printInterfaceD() {
          System.out.println("I am from D " + k);
     }
public void printB() {
         super.print();
         printInterfaceC();
         printInterfaceD();
    }

}


class Demonstration_910 {
      public static void main (String[] args)     {
               B b = new B();
               b.printB();
   }
}
```

Output:-

```
D:\VJ \jvm>javac Demonstration_910.java
D:\VJ \jvm>java Demonstration_910
I am from A 1000
I am from C 555
I am from D 666
```

Experiment No. 11

Aim: - Study of Packages with a program to demonstrate use of user defined packages.

Theory:-

A Java **package** is a mechanism for organizing Java classes. Packages are used in Java, in-order to avoid name conflicts and to control access of class, interface and enumeration etc. A package can be defined as a group of similar types of classes, interface, enumeration and **sub-packages**. We can define package using package keyword.

Built-in Package :-

Existing built-in java packages like java.lang, java.util, java.sql etc.

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.

The library contains components for managing input, database programming, and much much more.

The library is divided into **packages** and **classes**. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

To use a class or a package from the library, you need to use the import keyword:

Syntax
import *package.name.Class*; // Import a single class
import *package.name*.*; // Import the whole package

User-define package :-

Java package created by user to categorize classes, interface and enumeration.

To create your own package, you need to understand that Java use a file system directory to store them. Just like folders on your computer:

Example

└ — root

└ — mypack

└ — MyPackageClass.java

To create a package, use the package keyword:

MyPackageClass.java

```
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```

Save the file as MyPackageClass.java, and compile it:

C:\Users\Your Name>javac MyPackageClass.java

Then compile the package:

C:\Users\Your Name>javac -d . MyPackageClass.java

This forces the compiler to create the "mypack" package.

The -d keyword specifies the destination for where to save the class file. You can use any directory name, like c:/user (windows), or, if you want to keep the package within the same directory, you can use the dot sign ".", like in the example above.

Note: The package name should be written in lower case to avoid conflict with class names.

When we compiled the package in the example above, a new folder was created, called "mypack".

To run the MyPackageClass.java file, write the following:

C:\Users\Your Name>java mypack.MyPackageClass

The output will be:

This is my package!

Program: - WAP to find the cube of a number for various data types using package and then import and display the results.

//1.program to create package

```
package mathematics;
public class Mathmethods
{
public static float Cube(float n)
{
return(n*n*n);
}
public static int Cube(int n)
{
return(n*n*n);
}
public static double Cube(double n)
{
return(n*n*n);
}
public static long Cube(long n)
{
return(n*n*n);
}
}
```

//2. program to import

```
import mathematics.Mathmethods;
import java.util.Scanner;
class Cube
{
public static void main(String S[])
{
//int a=20;
Scanner m=new Scanner(System.in);
```

```
System.out.println("the given number is ");
int a=m.nextInt();
Mathmethods mm = new Mathmethods();
int b = Mathmethods.Cube(a);
System.out.println("cube is " +b);
}
}
```

Output:-
D:\VJ \jvm>javac -d . Mathmethods.java
D:\VJ \jvm>javac Cube.java
D:\ VJ \jvm>>java Cube
the given number is
2
cube is 8

Experiment No. 12

Aim: - Program on to demonstrate use of Exception handling by divide of zero situations.

Theory:-

Exception is a condition that is caused by run-time error in the program. In computer programming languages that do not support exception handling, errors must be checked and handled manually through the use of error codes, and so on. But this approach is cumbersome as well as troublesome. Java's exception handling avoids these problems and, in the process and brings run-time error management into the object-oriented world. C++ programming language supports exception handling but Java has enhanced some of its features.

As Java is strictly object oriented, an exception is also an object that describes an exceptional (that is, error) condition that has occurred in a piece of source code. When an exceptional condition arises in program, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Exceptions can be generated by the Java run-time system, or they can be manually generated by our code.

Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically used to report some error condition to the caller of a method.

The purpose of exception handling mechanism is to provide a means to detect and to report exceptional circumstances so that appropriate action can be taken.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).

Java try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

Syntax

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Consider the following example:

This will generate an error, because myNumbers[10] does not exist.

```
public class MyClass {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

The output will be something like this:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at MyClass.main(MyClass.java:4)
```

If an error occurs, we can use try...catch to catch the error and execute some code to handle it:

Example

```
public class MyClass {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

The output will be:

```
Something went wrong.
```

Finally

The `finally` statement lets you execute code, after `try...catch`, regardless of the result:

Example

```
public class MyClass {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

The output will be:

```
Something went wrong.
The 'try catch' is finished.
```

Program

```
import java.io.*;
import java.util.*;
class MyExceptions
{
void dzero( int a, int b)
{
try
{
int d=a/b;
System.out.println("\n Division= "+d);
}
catch(Exception e)
{
System.out.println(" Sorry Division by zero not possible");
}
}
}
class Exceptions
{
public static void main(String args[])
{
MyExceptions e=new MyExceptions();
Scanner sc=new Scanner(System.in);
System.out.println("\n Enter first number");
int n1=sc.nextInt();
System.out.println("\n Enter second number");
int n2=sc.nextInt();
e.dzero(n1,n2);
}
}
```

```
D:\VJ \jvm>javac Exceptions.java

D:\VJ \jvm>java Exceptions

 Enter first number
10
 Enter second number
0
 Sorry Division by zero not possible
```

Experiment No. 13

Aim: - Program to demonstrate Multithreading by implementing thread using runnable interface.

Theory:-

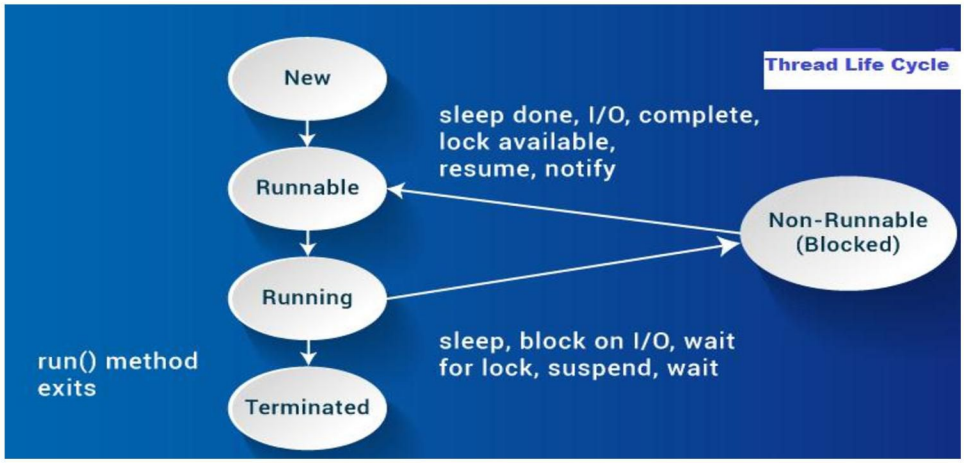
Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs.

The Java Thread Model

The Java run-time system depends on threads for many things. Threads reduce inefficiency by preventing the waste of CPU cycles.

Threads exist in several states. Following are those states:

- **New** – When we create an instance of Thread class, a thread is in a new state.
- **Runnable –** The Java thread is in running state.
- **Suspended** – A running thread can be **suspended**, which temporarily suspends its activity. A suspended thread can then be resumed, allowing it to pick up where it left off.
- **Blocked** – A java thread can be blocked when waiting for a resource.
- **Terminated** – A thread can be terminated, which halts its execution immediately at any given time. Once a thread is terminated, it cannot be resumed.



Java's multithreading system is built upon the Thread class, its methods, and its companion interface, **Runnable**. To create a new thread, your program will either extend **Thread** or **implement** the **Runnable interface**.

The Thread class defines several methods that help manage threads.The table below displays the same:

Method	Meaning
getName	Obtain thread's name
getPriority	Obtain thread's priority
isAlive	Determine if a thread is still running
join	Wait for a thread to terminate
run	Entry point for the thread
sleep	Suspend a thread for a period of time
start	Start a thread by calling its run method

Runnable Interface

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

```
public void run( )
```

Inside run( ), we will define the code that constitutes the new thread

Example:

```
public class MyClass implements Runnable {
public void run(){
System.out.println("MyClass running");
   }
}
```

To execute the run() method by a thread, pass an instance of MyClass to a Thread in its constructor(*A constructor in Java is a block of code similar to a method that's called when an instance of an object is created*). Here is how that is done:

```
Thread t1 = new Thread(new MyClass ());
t1.start();
```

When the thread is started it will call the run() method of the MyClass instance instead of executing its own run() method. The above example would print out the text "**MyClass running**".

Extending Java Thread

The second way to create a thread is to create a new class that extends Thread, then override the run() method and then to create an instance of that class. The run() method is what is executed by the thread after you call start(). Here is an example of creating a Java Thread subclass:

```
public class MyClass extends Thread {
     public void run(){
     System.out.println("MyClass running");
   }
}
```

To create and start the above thread you can do like this:

```
MyClass t1 = new MyClass ();
T1.start();
```

When the run() method executes it will print out the text "**MyClass running**".

Program

```
class MyThread implements Runnable
{
String message;
MyThread(String msg)
{
message = msg;
}
public void run()
{
for(int count=0;count<=5;count++)
{
try
{
System.out.println("Run method: " + message);
Thread.sleep(100);
}
catch (InterruptedException ie)
{
System.out.println("Exception in thread: "+ie.getMessage());
} } } }
public class MainThread
{
public static void main(String[] args)
{
MyThread obj1 = new MyThread("MyThread obj1");
MyThread obj2 = new MyThread("MyThread obj2");
Thread t1 = new Thread(obj1);
Thread t2 = new Thread(obj2);
t1.start();
t2.start();
}
}
```

OUTPUT:

```
D:\VJ \jvm>javac MainThread.java

D:\VJ \jvm>java MainThread
Run method: MyThread obj1
Run method: MyThread obj2
Run method: MyThread obj1
Run method: MyThread obj2
Run method: MyThread obj2
Run method: MyThread obj1
Run method: MyThread obj1
Run method: MyThread obj2
Run method: MyThread obj2
Run method: MyThread obj1
Run method: MyThread obj2
Run method: MyThread obj1
```

Experiment No. 14

Aim: - Study of Applet with a program to display various shapes in some different colors.

Theory:-

Java Applets

Applets are small Internet-based program written in Java, a programming language for the Web and can be downloaded by any computer. The applet is also capable of running in HTML. The applet is usually embedded in an HTML page on a Web site and can be executed from within a browser. After an applet arrives on the client, it has limited access to resources so that it can produce a graphical user interface and run complex computations without introducing the risks of viruses or data security breaching.

This applet begins with two import statements. The first import statement is for the Abstract Window Toolkit (AWT) classes. Applets interact with the user through the AWT and not through the console-based I/O classes. An applet must be a subclass of the java.applet.Applet class. The Applet class provides the standard interface between the applet and the browser environment. Swing provides a special subclass of the Applet class called javax.swing.JApplet. The JApplet class should be used for all applets that use Swing components to construct their graphical user interfaces (GUIs). The browser's Java Plug-in software manages the lifecycle of an Applet. The applet in Java can appear in a frame of the web page, a new application window, Sun's AppletViewer, or a stand-alone tool for testing them. They were introduced in the first version of the Java language, which was introduced in the year 1995.

Java applets have been mentioned here for information purposes, but you also need to know that Java applets are deprecated from Java in 2017 and have been completely removed from Java SE 11 (18.9) released in September 2018.

Benefits of Java Applets

- They are very secure.
- It works at client side so less response time.
- Applets can be executed by browsers running under different platforms.

One disadvantage of Applets is that plugins are required at the client browser for executing applets.

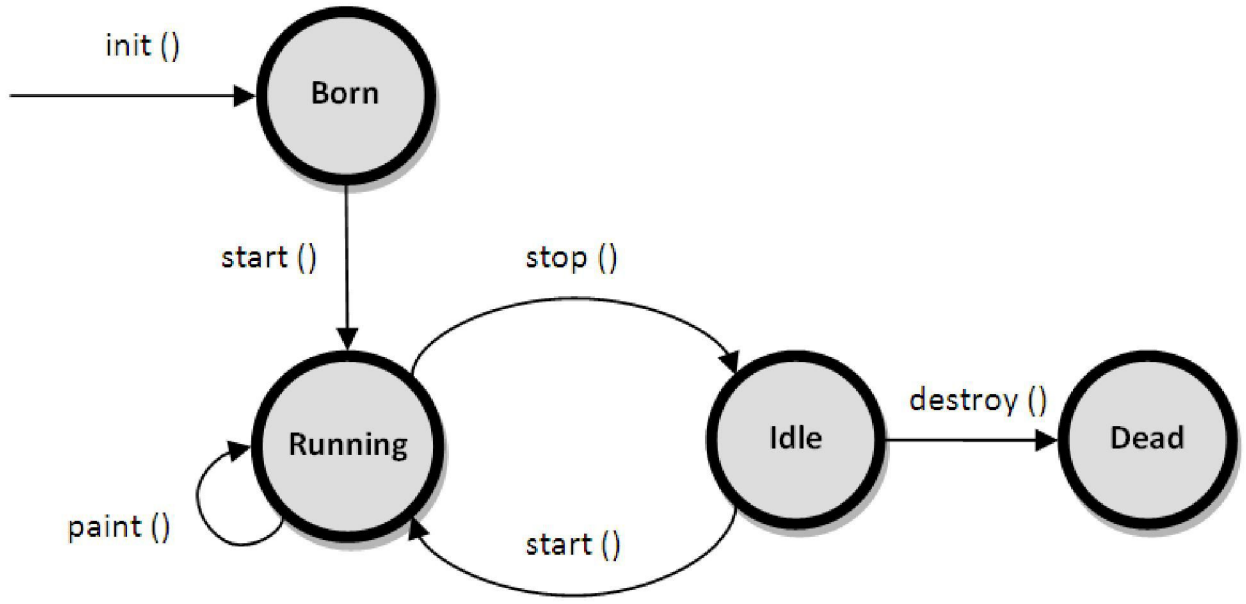
Browser Responsibilities for Applet Life Cycle

Applet life cycle methods are callback methods because they are called implicitly by the browser for the smooth execution of the applet. The browser should provide an environment known as a container for the execution of the applet. Following are the responsibilities of the browser.

- For the smooth execution, it should call the callback methods at appropriate times.
- It is responsible to maintain the Applet Life Cycle.
- It should have the capability to communicate between applets, applet to JavaScript and HTML, applet to browser, etc.

Applet Life Cycle

Even though, the methods are called automatically by the browser, the programmer should know well when they are called and what he can do with the methods. Following is the schematic representation of the methods.



It is important to understand the order in which the various methods shown in the above image are called. When an applet begins, the following methods are called, in this sequence:

1. init( )
2. start( )
3. paint( )

When an applet is terminated, the following sequence of method calls takes place:

1. stop( )
2. destroy( )

Let's look more closely at these methods.

1. **init( ) :** The **init( )** method is the first method to be called. This is where you should initialize variables. This method is called **only once** during the run time of your applet.
2. **start( ) :** The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been stopped. Note that **init( )** is called once i.e. when the first time an applet is loaded whereas **start( )** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.
3. **paint( ) :** The **paint( )** method is called each time an AWT-based applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored.
4. **stop( ) :** The **stop( )** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop( )** is called, the applet is probably running. You should use **stop( )** to suspend threads that don't need to run when the applet is not visible. You can restart them when **start( )** is called if the user returns to the page.
5. **destroy( ) :** The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop( )** method is always called before **destroy( )**

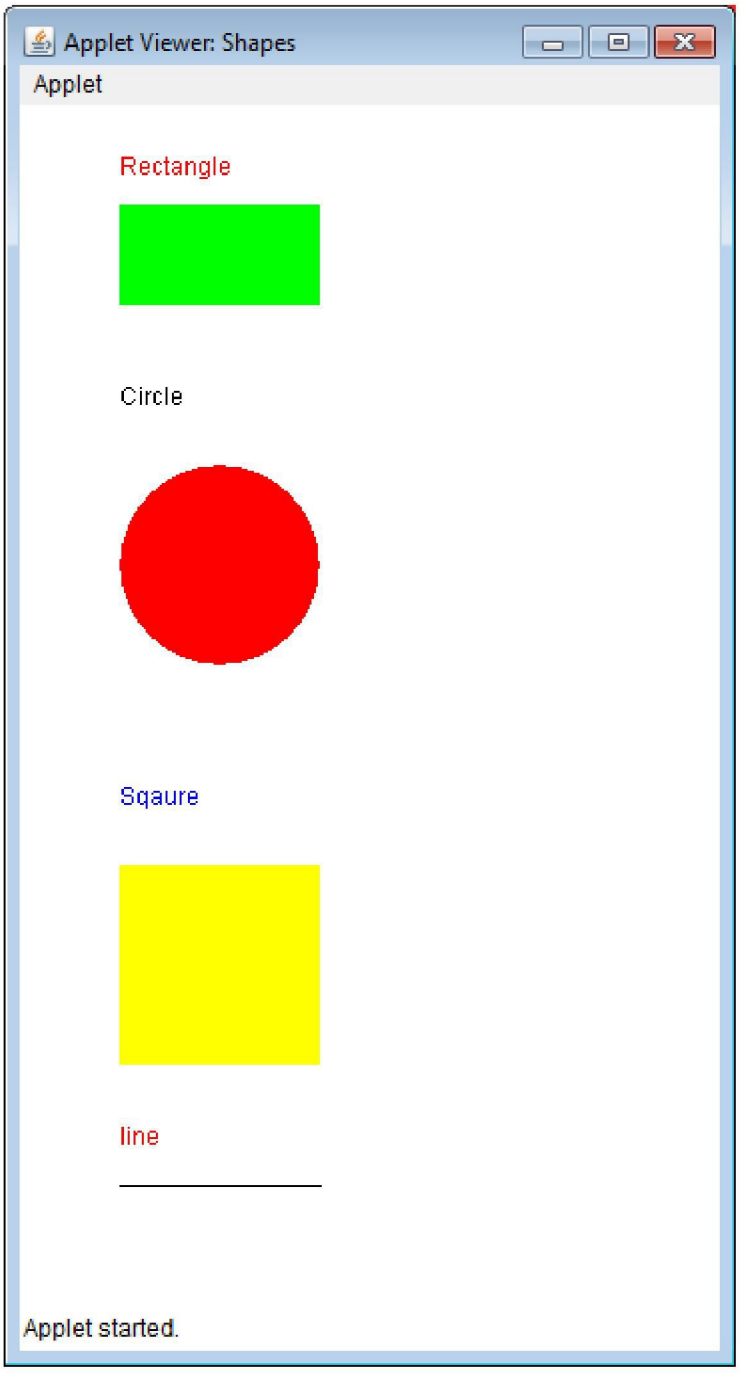
Program

```
import java.applet.*;
import java.awt.*;
//<applet code="Shapes" height="600" width="350"></applet>
public class Shapes extends Applet
{
public void paint(Graphics g)
{
g.setColor(Color.red);
g.drawString("Rectangle",50,35);
g.setColor(Color.green);
g.fillRect(50,50,100,50);
g.setColor(Color.black);
g.drawString("Circle",50,150);
g.setColor(Color.red);
g.fillOval(50,180,100,100);
g.setColor(Color.blue);
g.drawString("Sqaure",50,350);
g.setColor(Color.yellow);
g.fillRect(50,380,100,100);
g.setColor(Color.red);
g.drawString("line",50,520);
g.setColor(Color.black);
g.drawLine(50,540,150,540);
}
}
```

OUTPUT:

D:\VJ \jvm>javac Shapes.java

D:\VJ \jvm>appletviewer Shapes.java


Applet Viewer: Shapes
Applet
Rectangle
Circle
Sqaure
line
Applet started.

Experiment No. 15

Aim: - Java program for Bouncing of a Ball using applet.

Theory:-

Throwback of making GUI application:

Java was launched on 23-Jan-1996(JDK 1.0) and at that time it only supported CUI(Character User Interface) application. But in 1996 VB(Visual Basic) of Microsoft was preferred for GUI programming. So the Java developers in hurry(i.e within 7 days) have given the support for GUI from Operating System(OS). Now, the components like button,etc. were platform-dependent(i.e in each platform there will be different size, shape button). But they did the intersection of such components from all platforms and gave a small library which contains these intersections and it is available in AWT(Abstract Window Toolkit) technology but it doesn't have advanced features like dialogue box, etc.

Important points :

1. All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class.
2. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
3. In general, execution of an applet does not begin at main() method.
4. Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

There are **two** standard ways in which you can run an applet :

1. Executing the applet within a Java-compatible web browser.
2. Using an applet viewer, such as the standard tool, applet-viewer. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.

Each of these methods is described next.

1. **Using java enabled web browser :** To execute an applet in a web browser we have to write a short HTML text file that contains a tag that loads the applet. We can use APPLET or OBJECT tag for this purpose. Using APPLET, here is the HTML file that executes HelloWorld :

```
<applet code="HelloWorld" width=200 height=60>
</applet>
```

The width and height statements specify the dimensions of the display area used by the applet. The APPLET tag contains several other options. After you create this html file, you can use it to execute the applet.

1. **Using appletviewer :** This is the easiest way to run an applet. To execute HelloWorld with an applet viewer, you may also execute the HTML file shown earlier. For example, if the preceding HTML file is saved with RunHelloWorld.html,then the following command line will run HelloWorld :

```
appletviewer RunHelloWorld.html
```

2. **appletviewer with java source file :** If you include a comment at the head of your Java source code file that contains the APPLET tag then your code is documented with a prototype of the necessary HTML statements, and you can run your compiled applet merely by starting the applet viewer with your Java source code file. If you use this method, the HelloWorld source file looks like this :

```
// A Hello World Applet
// Save file as HelloWorld.java

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

/*
<applet code="HelloWorld" width=200 height=60>
</applet>
*/

// HelloWorld class extends Applet
public class HelloWorld extends Applet
{
    // Overriding paint() method
    @Override
    public void paint(Graphics g)
    {
        g.drawString("Hello World", 20, 20);
    }

}
```

With this approach, first compile HelloWorld.java file and then simply run below command to run applet :

```
appletviewer HelloWorld.java
```

Program

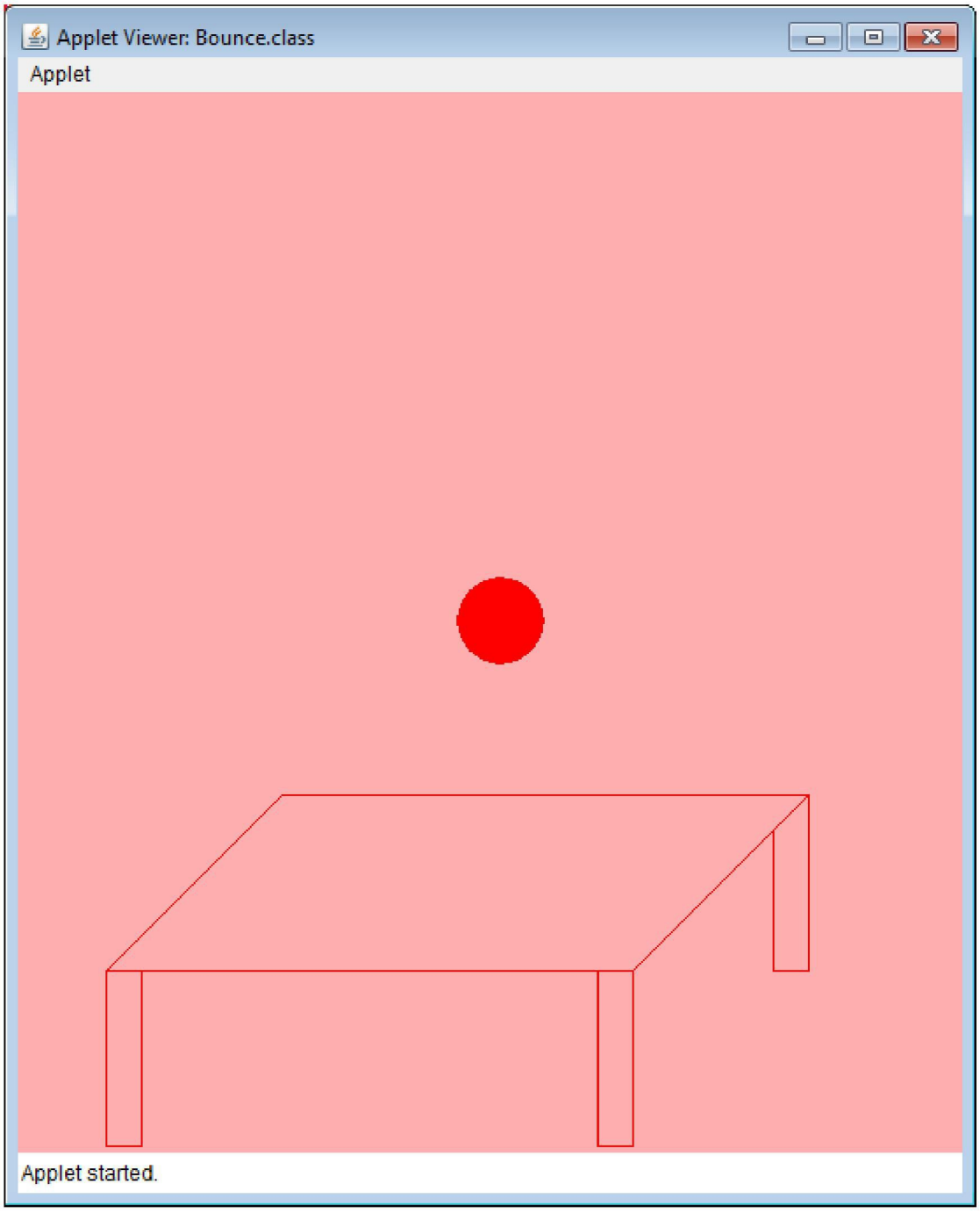
```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
/*<applet code= "Bounce.class" height=900 width=900> </applet>*/
 class Ball
{
int x,y,radius,dx,dy;
Color BallColor;
 public Ball(int x,int y,int radius,int dx,int dy,Color bColor)
 {
this.x=x;
this.y=y;
this.radius=radius;
this.dx=dx;
this.dy=dy;
BallColor=bColor;
} }
 public class Bounce extends Applet implements Runnable
{
Ball redBall;
 public void init()
 {
redBall=new Ball(250,80,50,2,4,Color.red);
Thread t=new Thread(this);
 t.start();
}
 public void paint(Graphics g)
{
g.setColor(redBall.BallColor);
 setBackground(Color.pink);
//g.setcolor(redBall.BallColor);
```

```
g.fillOval(redBall.x, redBall.y, redBall.radius,redBall.radius);
g.drawLine(150,400,50,500);
g.drawLine(150,400,450,400);
g.drawLine(50,500,350,500);
g.drawLine(450,400,350,500);
g.drawRect(50,500,20,100);
 g.drawRect(330,500,20,100);
g.drawLine(450,400,450,500);
g.drawLine(430,500,450,500);
g.drawLine(430,500,430,420);
}
public void run()
{ while(true)
 {
try
 {
displacementOperation(redBall);
Thread.sleep(20);
repaint();
}
catch(Exception e)
{
}
 }
 }
public void displacementOperation(Ball ball)
{
if(ball.y >= 400 || ball.y <= 0)
{
ball.dy=-ball.dy; } ball.y=ball.y+ball.dy;
 }
}
```

OUTPUT:

```
D:\VJ \jvm>javac Bounce.java

D:\VJ \jvm>appletviewer Bounce.java
```



Experiment No. 16

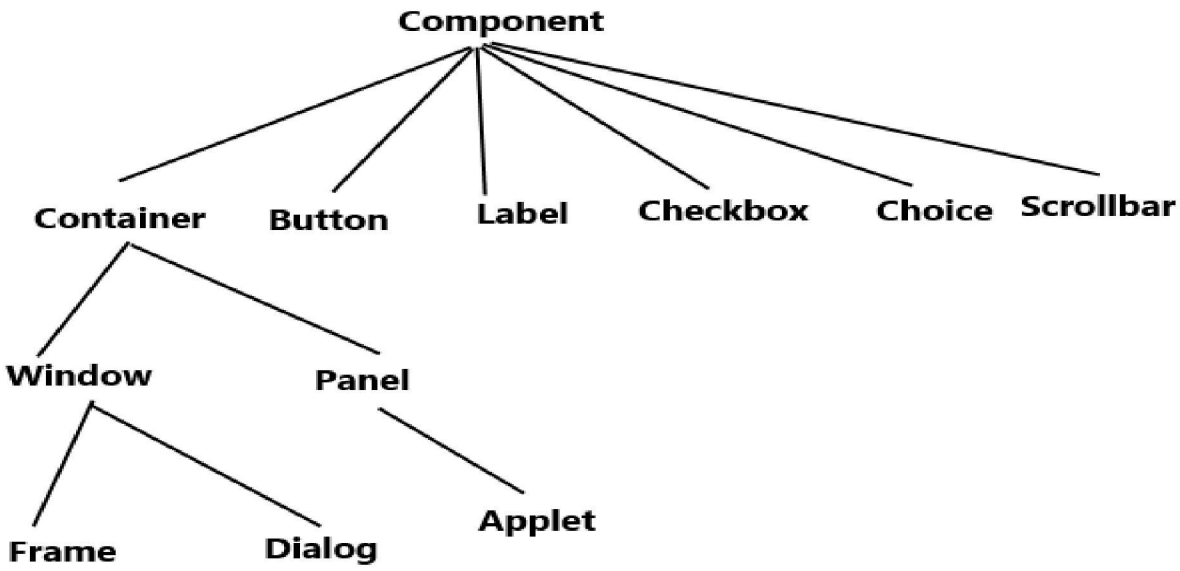
Aim: - Study of Abstract Window toolkit with a Program to display simple Textfields & Button controls on Frame. Display First Name/Last name on textField according to event called(Botton Clicked by User.)

Theory:-

AWT stands for **Abstract Window Toolkit**. It is a platform dependent API for creating Graphical User Interface (GUI) for java programs.

Why AWT is platform dependent? Java AWT calls native platform (Operating systems) subroutine for creating components such as textbox, checkbox, button etc. For example an AWT GUI having a button would have a different look and feel across platforms like windows, Mac OS & Unix, this is because these platforms have different look and feel for their native buttons and AWT directly calls their native subroutine that creates the button. In simple, an application build on AWT would look like a windows application when it runs on Windows, but the same application would look like a Mac application when runs on Mac OS.

AWT is rarely used now days because of its platform dependent and heavy-weight nature. AWT components are considered heavy weight because they are being generated by underlying operating system (OS). For example if you are instantiating a text box in AWT that means you are actually asking OS to create a text box for you.



Program

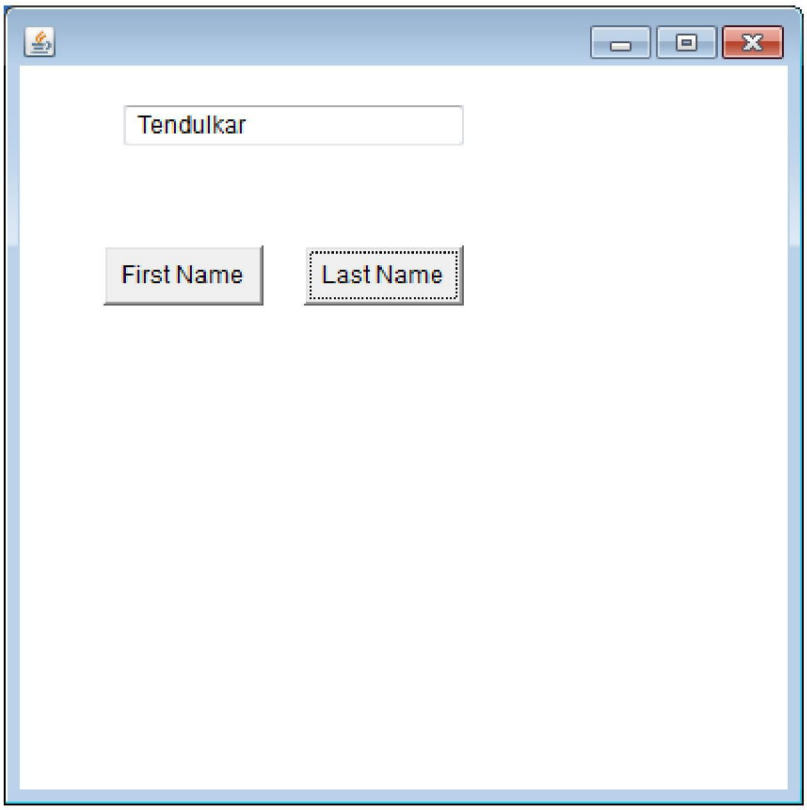
```
import java.awt.*;
import java.awt.event.*;
class AEvent_Second extends Frame implements ActionListener{
TextField tf;
Button b1,b2,b3;
AEvent_Second(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
b1=new Button("First Name");
b1.setBounds(50,120,80,30);
b2=new Button("Last Name");
b2.setBounds(150,120,80,30);
//register listener
b1.addActionListener(this);//passing current instance
b2.addActionListener(this);
//add components and set size, layout and visibility
add(b1);
add(b2);
//add(b3);
add(tf);
setSize(400,400);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
if(e.getSource()==b1)
{
tf.setText("Sachin ");
}
if(e.getSource()==b2)
```

```
{
tf.setText(" Tendulkar");
}
}
public static void main(String args[]){
new AEvent_Second();
}
}
```

OUTPUT:

D:\VJ \jvm>javac AEvent_Second.java

D:\VJ \jvm>java AEvent_Second



Experiment No. 17

Aim: - Write AWT Program to create simple arithmetic calculator by using Event Handling.

Theory:-

Features of AWT in Java

- AWT is a set of native user interface components
- It is based upon a robust event-handling model
- It provides Graphics and imaging tools, such as shape, color, and font classes
- AWT also avails layout managers which helps in increasing the flexibility of the window layouts
- Data transfer classes are also a part of AWT that helps in cut-and-paste through the native platform clipboard
- Supports a wide range of libraries that are necessary for creating graphics for gaming products, banking services, educational purposes, etc.

AWT UI Aspects

Any UI will be made of three entities:

- **UI elements**: These refers to the core visual elements which are visible to the user and used for interacting with the application. AWT in Java provides a comprehensive list of widely used and common elements.
- **Layouts**: These define how UI elements will be organized on the screen and provide the final look and feel to the GUI.
- **Behavior**: These define the events which should occur when a user interacts with UI elements.

Components and containers

All the elements like buttons, text fields, scrollbars etc are known as components. In AWT we have classes for each component as shown in the above diagram. To have everything placed on a screen to a particular position, we have to add them to a container. A container is like a screen wherein we are placing components like buttons, text fields, checkbox etc. In short a container contains and controls the layout of components. A container itself is a component (shown in the above hierarchy diagram) thus we can add a container inside container.

Types of containers:

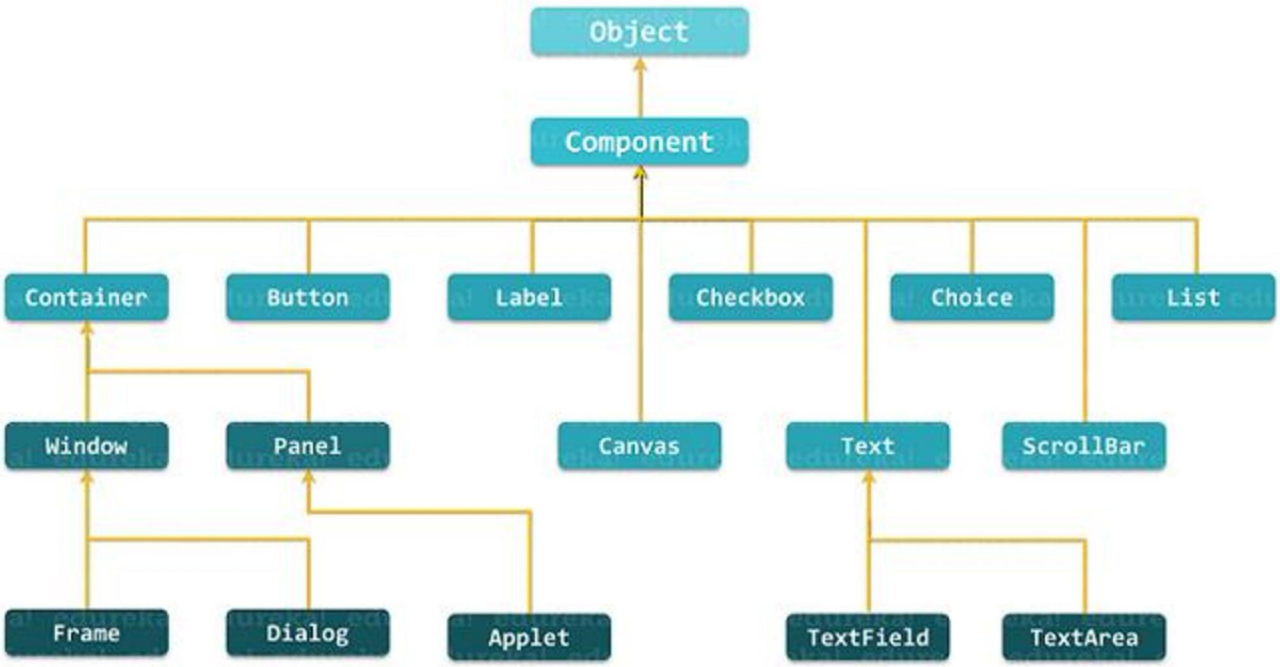
As explained above, a container is a place wherein we add components like text field, button, checkbox etc. There are four types of containers available in AWT: Window, Frame, Dialog and Panel. As shown in the hierarchy diagram above, Frame and Dialog are subclasses of Window class.

Window: An instance of the Window class has no border and no title

Dialog: Dialog class has border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.

Panel: Panel does not contain title bar, menu bar or border. It is a generic container for holding components. An instance of the Panel class provides a container to which to add components.

Frame: A frame has title, border and menu bars. It can contain several components like buttons, text fields, scrollbars etc. This is most widely used container while developing an application in AWT.



Program

```
import java.awt.*;
import java.awt.event.*;
class AEvent_Second extends Frame implements ActionListener{
TextField tf;
Button b1,b2,b3;
AEvent_Second(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
b1=new Button("First Name");
b1.setBounds(50,120,80,30);
b2=new Button("Last Name");
b2.setBounds(150,120,80,30);
//register listener
b1.addActionListener(this);//passing current instance
b2.addActionListener(this);
//add components and set size, layout and visibility
add(b1);
add(b2);
//add(b3);
add(tf);
setSize(400,400);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
if(e.getSource()==b1)
{
tf.setText("Sachin ");
}
if(e.getSource()==b2)
```

```
{
tf.setText(" Tendulkar");
}
}
public static void main(String args[]){
new AEvent_Second();
}
}
```

OUTPUT:

```
D:\VJ \jvm>javac Acalci.java

D:\VJ \jvm>java Acalci
```