Mahatma Gandhi Mission

Jawaharlal Nehru Engineering College

Aurangabad, Maharashtra

Affiliated to Dr. B. A. Technological University, Lonere

NAAC 'A' Grade, ISO 9001:2015, 14001:2015 Certified, AICTE Approved.

Second Year B. Tech

Department of Information Technology

Lab Book

BTITL 309B: Programming in JAVA lab

Name: _____

Class:_____ Roll No: _____ Year: _____

Exam No.:_____

Mahatma Gandhi Mission

Jawaharlal Nehru Engineering College

Aurangabad, Maharashtra

Affiliated to Dr. B. A. Technological University, Lonere

NAAC 'A' Grade, ISO 9001:2015, 14001:2015 Certified, AICTE Approved.

Second Year B. Tech

Department of Information Technology

Lab Book

BTITL 309B: Programming in JAVA lab

Prepared by
Prof. Mr. Yogesh Tayade
Assistant Professor
Lab Incharge

Reviewed by
Dr. S. C. Tamane
Associate Professor
Head of Department

Approved by
Dr. H. H. Shinde
Principal

Vision of Information Technology Department:

To develop expertise of budding technocrats by imparting technical knowledge and human value-based education.

Mission of Information Technology Department:

A. Equipping the students with technical skills, soft skills and professional attitude.

B. Providing the state of art facilities to the students to excel as competent professionals, entrepreneurs and researchers.

Programme Educational Objectives:

PEO1. The graduates will utilize their **expertise** in IT industry and solve industry technological problems.
PEO2. Graduates should excel in **engineering positions** in industry and other organizations that emphasize design & implementation of IT applications.
PEO3. Graduates will be **innovators & professionals** in technology development, deployment & system implementation.
PEO4. Graduates will be pioneers in engineering, engineering management, research and **higher education**.
PEO5. Graduates will be good citizens & cultured human being with full appreciation of importance of IT **professional ethical & social** responsibilities.

Program specific outcomes

- PSO1. An ability to design, develop and implement computer programs in the areas related to Algorithms, Multimedia, Website Design, System Software, DBMS and Networking.
- PSO2. Develop software systems that would perform tasks related to Research, Education and Training and/or E governance.
- PSO3. Design, develop, test and maintain application software that would perform tasks related to information management and mobiles by utilizing new technologies to an individual or organizations.

Lab outcomes: After the completion of this course students will be able to,

LO1 Know the structure and object-oriented model of the Java programming language.
LO2 Use the Java programming language methods, etc. to solve ealtime problem statements.
LO3 invoking methods, using class libDevelop application in the Java programming language aries, etc.

Mandatory instructions for students:

1. Students should report to the concerned labs as per the given timetable.
2. Students should make an entry in the log book whenever they enter the labs during practical or for their own personal work.
3. When the experiment is completed, students should shut down the computers and make the counter entry in the logbook.
4. Any damage to the lab computers will be viewed seriously.
5. Students should not leave the lab without concerned faculty's permission.

Mahatma Gandhi Mission

Jawaharlal Nehru Engineering College

N-6, CIDCO Aurangabad – 431003

INDEX OF EXPERIMENTS

Name of student: _____

Roll No: _____ PNR No: _____ Batch: _____

Sr. No.	Practical details	Date of performance	Date of evaluation	Marks/ Grade	Sign

Submission Details:

Date: Sign of Faculty Sign of HOD

Exercise - 1

Aim: To study the working environment of java.

Theory:

Introduction

Java programming language is an *interpreted* programming language, that is, when the source code is compiled into binary file, it needs an interpreter called Java Virtual Machine (JVM) to run it.
Java compiler is called javac.exe, and interpreter is called java.exe.

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

Let's look at how to save the file, compile and run the program. Please follow the steps given below:

Open notepad and add the code as above.
Save the file as: MyFirstJavaProgram.java.
Open a command prompt window and go to the directory where you saved the class. Assume it's C:
Type ' javac MyFirstJavaProgram.java ' and press enter to compile your code. If there are no errocode, the command prompt will take you to the next line(Assumption : The path variable is set).
Now, type ' java MyFirstJavaProgram ' to run your program.
You will be able to see ' Hello World ' printed on the window.

```
 C :> javac MyFirstJavaProgram.java
 C :> java MyFirstJavaProgram
    HelloWorld
```

Program using command line arguments:-

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

Program takes input from command line.
For e.g.

```
C :> javac Commd.java
C :> java Commd Hello World
        Hello World
```

Assignents:

Wap to print your Biodata?
Wap to add,substract,devide,multiply two numbers using command line arguments?
Wap to print your Biodata using command line arguments?
Wap to find out simple interest and compound interest?

WORKSPACE:

Exercise - 2

Aim: To study Decision Control structures and loop control structures in java

Theory:

Loop:-It is used when we need to execute a block of code several number of times, and is often referred to as a loop.

Java has very flexible three looping mechanisms. You can use one of the following three loops:

while Loop
do...while Loop
for Loop

The while Loop:

A while loop is a control structure that allows you to repeat a task a certain number of times.

Syntax:

The syntax of a while loop is: while(Boolean_expression)

```
{
//Statements
}
```

When executing, if the boolean_expression result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Here, key point of the while loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

```
public class Test
{
        public static void main(String args[])
        {
                 int x = 10;
                while( x < 20 )
                {
                        System.out.print("value of x : " + x );
                        x++;
                        System.out.print("\n");
                }
        }
}
```

This would produce the following result:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
```

value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19

The do...while Loop:

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

The syntax of a do...while loop is: do

```
{
//Statements
}while(Boolean_expression);
```

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example:

```
public  class Test
{
        public  static  void main(String args[])
        {
                int x =10;
                do
                {
                        System.out.print("value of x : "+ x );
                        x++;
                        System.out.print("\n");
                }while( x <20);
        }
}
```

This would produce the following result:

value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19

The for Loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

Syntax:
The syntax of a for loop is: for(initialization;Boolean_expression; update)
{
//Statements
}

Here is the flow of control in a for loop:

The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression. The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

Example:

```
public class Test
{
        Public static void main(String args[])
        {
                for(int x =10; x <20; x = x+1)
                {
                        System.out.print("value of x : "+ x );
                        System.out.print("\n");
                }
        }
}
```

This would produce the following result:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

Decision Making:

Here are two types of decision making statements in Java. They are:

if statements
switch statements

The if Statement:

An if statement consists of a Boolean expression followed by one or more statements. Syntax:
The syntax of an if statement is:

```
if(Boolean_expression)
{
//Statements will execute if the Boolean expression is true
}
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Example:

```
public class Test
```

```
{
        public static void main(String args[])
        {
                int x =10;
                if( x <20)
                {
                        System.out.print("This is if statement");
                }
        }
}
```

This would produce the following result:

```
This is if statement
```

The if...else Statement:

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

Syntax:

The syntax of an if...else is:

```
 if(Boolean_expression)
{
//Executes when the Boolean expression is true
}
else
{
//Executes when the Boolean expression is false
}
```

Example:

```
public class Test
{
        public static void main(String args[])
        {
                int x =30;
                if( x <20)
                {
                        System.out.print("This is if statement");
                }
                else
                {
                        System.out.print("This is else statement");
                }
        }
}
```

This would produce the following result:

```
This is else statement
```

The if...else Statement:

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind.

An if can have zero or one else's and it must come after any else if's.
An if can have zero to many else if's and they must come before the else.
Once an else if succeeds, none of the remaining else if's or else's will be tested.

Syntax:
The syntax of an if...else is: if(Boolean_expression1)
{
//Executes when the Boolean expression 1 is true
}
else if(Boolean_expression2){
//Executes when the Boolean expression 2 is true
}
else if(Boolean_expression3){
//Executes when the Boolean expression 3 is true
}
else{
//Executes when the none of the above condition is true.
}
Example:

```
public class Test
{
	public static void main(String args[])
	{
		int x =30;
		if( x ==10)
		{
			System.out.print("Value of X is 10");
		}
		else if( x ==20)
		{
			System.out.print("Value of X is 20");
		}
		else if( x ==30)
		{
			System.out.print("Value of X is 30");
		}
		else
		{
			System.out.print("This is else statement");
		}
	}
```

```
}
```
This would produce the following result:
Value of X is 30

Nested if...else Statement:

It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

Syntax:
The syntax for a nested if...else is as follows:

```
if(Boolean_expression1){
//Executes when the Boolean expression 1 is true
if(Boolean_expression2){
//Executes when the Boolean expression 2 is true
}
}
```

You can nest else if...else in the similar way as we have nested if statement.
Example:

```
public class Test
{
        public static void main(String args[])
        {
                int x =30;
                int y =10;
                if( x ==30)
                {
                        if( y ==10)
                        {
                                System.out.print("X = 30 and Y = 10");
                        }
                }
        }
}
```

This would produce the following result:
X =30 and Y =10

The switch Statement:

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

The syntax of enhanced for loop is:

```
switch(expression)
{
case value :
```

```
        //Statements
        break;  //optional
case value :
        //Statements
         break; //optional
//You can have any number of case statements.
default:
        //Optional
        //Statements
}
```

The following rules apply to a switch statement:

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Example:

```
public class Test
{
        public static void main(String args[])
        {
                char grade ='C';
                switch(grade)
                {
                        case'A':
                                System.out.println("Excellent!");
                                break;
                        case'B':
                        case'C':
                                System.out.println("Well done");
                                break;
                        case'D':
                                System.out.println("You passed");
                                break;
                        case'F':
```

```
                        System.out.println("Better try again");
                        break;
                default:
                        System.out.println("Invalid grade");
            }
        System.out.println("Your grade is "+ grade);
        }
}
```

Compile and run above program using various command line arguments. This would produce the following result:

```
Welldone
Your grade is a C
```

The break Keyword:

The break keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop: break;

Example:

```
public class Test
{
        public static void main(String args[])
        {
                int[] numbers ={10,20,30,40,50};
                for(int x : numbers )
                {
                        if( x ==30)
                        {
                        break;
                        }
                System.out.print( x );
                System.out.print("\n");
                }
        }
}
```

This would produce the following result:

```
10
20
```

The continue Keyword:

The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.
Syntax:
The syntax of a continue is a single statement inside any loop: continue;
Example:

```
public class Test
{
        public static void main(String args[])
        {
                int[] numbers ={10,20,30,40,50};

                for(int x : numbers )
                {
                        if( x ==30)
                        {
                        continue;
                        }
                System.out.print( x );
                System.out.print("\n");
                }
        }
}
```

This would produce the following result:

```
10
20
40
50
```

Exercise:-

1. Wap to find out Factorial of a number using For loop, While loop, Do while loop.
2. Wap to find out square root of a number without using any library function?
3. Wap to find out wheather a number is palindrome or not?
4. Write program to print the kth digit from last. e.g. input 23617 and k=4 output 3.
5. Write program to find sum of all digits. Input 23617 output 2+3+6+1+7=19.
6. Write program, which will find sum of product to consecutive digits. e.g. when the input
7. is 23145 the output is 2x3 + 3x1 + 1x4 + 4x5 = 33?
8. Write program, which reads two number (assume that both have same number of digits).?
9. The program outputs the sum of product of corresponding digits. Input 327 and 539 output 3x5+2x3+7x9=84 ?
10. Write program to find sum of even digits. Input 23617 output 2+6=8 ?
11. Write program to find number of digits. Input 423 output 3. Input 21151 output 5 ?
12. Write program to print the last even digit. e.g. input 23613 output 6 ?
13. Write program to print the second last even digit. e.g. input 23863 output 8 (do not use
14. 'if'). Input 325145761 output 4. *Hint: use two loops] ?
15. Read a number. Do half of number after last odd digit. Input 3 times. Input 61389426 output 184167639 (61389213*3). Input 87 output 261. Input 78 output 222 (74*3) ?
16. Find sum of numbers formed by exchanging consecutive digits. In above 42+14+51=107?

WORKSPACE:

Exercise - 3

Aim: To study Arrays in java

Theory:

Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

dataType[] arrayRefVar; // preferred way. or
dataType arrayRefVar[]; // works but not preferred way.

Note: The style dataType[] arrayRefVar is preferred. The style dataType arrayRefVar[] comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

Creating Arrays:

You can create an array by using the new operator with the following syntax: arrayRefVar = new dataType[arraySize];
The above statement does two things:

1. It creates an array using new dataType[arraySize];
2. It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:
dataType[] arrayRefVar = new dataType[arraySize];

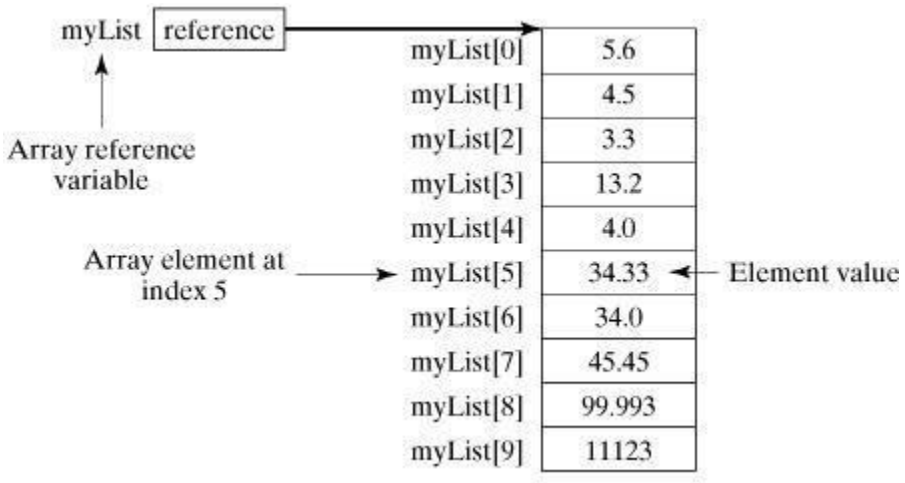
Alternatively you can create arrays as follows:
dataType[] arrayRefVar = {value0, value1, ..., valuek};
The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

double[] myList = new double[10];

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays:

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example:-
searching maximum no in an arrary:

```
public class TestArray
{
	public static void main(String[] args)
	{
		double[] myList ={1.9,2.9,3.4,3.5};
		// Print all the array elements
		 for(int i =0; i < myList.length; i++)
		{
			System.out.println(myList[i]+" ");
		}
		// Summing all elements double
		total =0;
		for(int i =0; i < myList.length; i++)
		{
			total += myList[i];
		}
		System.out.println("Total is "+ total);
		// Finding the largest element
```

```
                double max = myList[0];
                for(int i =1; i < myList.length; i++)
                {
                        if(myList[i]> max)
                        {
                                max = myList[i];
                        }
                }
                System.out.println("Max is "+ max);
        }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

Example 2:
Matrix Addition

```
public class Add
{
        public static void main(String args[])
        {
                int [][] x={ {1,2,3},{4,5,6},{7,8,9} };
                int [][] y={ {11,12,13},{14,15,16},{17,18,19} };
                int [][] z=new int[3][3];
                for(int i=0; i<3; i++)
                {
                        for(int j=0; j<3; j++)
                        {
                                z[i][j]=x[i][j]+y[i][j];
                        }
                }
                for(int i=0; i<3; i++)
                {
                        for(int j=0; j<3; j++)
                        {
                                System.out.print(z[i][j] +" ");
                        }
                        System.out.print("\n");
                }
        }
}
```

Exercise-

1. Insert 10 numbers into stack using push operation then delete 3 elements using pop operation and display rest?
2. Find smallest number in an array.
3. Find largest number in an array.
4. Count even numbers in an array.
5. Count occurrence of a given number in an array.
6. Input two arrays and merge them in a new array in ascending order.
7. Find Addition of two 3X3 matrices.
8. Find Multiplication of two 3X3 matrices.
9. Find Transpose of a given matrices.

WORKSPACE:

Exercise - 4

Aim: To study classes and objects in JAVA

Theory:

Object - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.

Class - A class can be defined as a template/blue print that describes the behaviors/states that object of it. A class is a blue print from which individual objects are created.

Class:-

```
public class Dog
{
        String breed;
        int age;
        String color;
        void barking()
        {
        }
        void hungry()
        {
        }
        void sleeping()
        {
        }
}
```

A class can contain any of the following variable types.

Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
Instance variables: Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
Class variables: Class variables are variables declared with in a class, outside any method, with the static keyword.
A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Constructors:

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
public class Puppy
{
        public Puppy()
        {
                //Default constructor
        }
        public Puppy(String name)
        {
                // This constructor has one parameter, name.
        }
}
```

Creating an Object:

As mentioned previously, a class provides the blueprints for objects. So basically an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

Declaration: A variable declaration with a variable name with an object type.
Instantiation: The 'new' key word is used to create the object.
Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
public class Puppy
{
        public Puppy(String name)
        {
                // This constructor has one parameter, name.
                System.out.println("Passed Name is :"+ name );
        }
        public static void main(String[]args)
        {
                // Following statement would create an object myPuppy
                Puppy myPuppy =new Puppy("tommy");
        }
}
```

If we compile and run the above program, then it would produce the following result:

```
Passed Name is : tommy
```

Accessing Instance Variables and Methods:

Instance variables and methods are accessed via created objects. To access an instance variable the fully qualified path should be as follows:

```
/* First create an object */
ObjectReference=new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

Example:
This example explains how to access instance variables and methods of a class:

```
public class Puppy
{
        int puppyAge;

        public Puppy(String name)
        {
                // This constructor has one parameter, name.
                System.out.println("Passed Name is :"+ name );
        }
        public void setAge(int age )
        {
                puppyAge = age;
        }

        public int getAge()
        {
                System.out.println("Puppy's age is :"+ puppyAge );
                return puppyAge;
        }
        public static void main(String[] args)
        {
                /* Object creation */
                Puppy myPuppy =newPuppy("tommy");

                /* Call class method to set puppy's age */
                myPuppy.setAge(2);

                /* Call another class method to get puppy's age */
                 myPuppy.getAge();
```

```
        /* You can access instance variable as follows as well */
        System.out.println("Variable Value : "+ myPuppy.puppyAge );
    }
}
```

If we compile and run the above program, then it would produce the following result:

```
Passed Name is : tommy
Puppy's age is : 2
Variable Value : 2
```

Exercise:-

1. Wap to find out area of a triangle having three sides using class?
2. Wap to find out volume of a box using class ?
3. Wap to find out area and perimeter of a rectangle ?
4. Wap to insert 10 no into stack using push operation and delete 2 elements using pop and display rest using class ?
5. Wap to find out addition,devision,multiplication,substraction and modulus of two objects using class ?
6. Wap to display the order in which constructors and destructors are invoked during runtime in class ?
7. Wap which specify the importance of using access specifier such as public and private in class?
8. Wap to implement given using class:- Queue, Linked List, Heap

WORKSPACE:

Exercise - 5

Aim: To study methods, method overloading , constructors in java

Theory:

Method overloading: Method having same name but different parameter.
Constructor overloading: Constructor having different parameters.
Example:-
Program explains the concept of method overloading and constructor overloading.

```
 class Cs
{
        int p,q;
        public Cs()
        {
        }
        public Cs(int x, int y)
        {
                p=x;
                q=y;
        }
        public int add(int i, int j)
        {
                return (i+j);
        }
        public int add(int i, int j, int k)
        {
                return (i+j+k);
        }
        public float add(float f1, float f2)
        {
                return (f1+f2);
        }
        public void printData()
        {
                System.out.print("p = "+p);
                System.out.println(" q = "+q);
        }
}
class Hari
{
        public static void main(String args[ ])
        {
                int x=2, y=3, z=4;
                Cs c=new Cs();
                Cs c1=new Cs(x, z);
                c1.printData();
```

```
                float m=7.2, n=5.2;
                int k=c.add(x,y);
                int t=c.add(x,y,z);
                float ft=c.add(m, n);
                System.out.println("k = "+k);
                System.out.println("t = "+t);
                System.out.println("ft = "+ft);
        }
}
```

Output:
p=2
q=4
k=5
t=9
ft=12.400000

Exercise:

1. Wap to find out area of a triangle,rectangle,square and circle using method overloading and constructor overloading.
2. Write a JAVA program which contains a method square() such that square(3) returns 9, square(0.2) returns 0.04.
3. Write a JAVA program which contains a method cube() such that cube(3) returns 27, cube(0.2) returns 0.008.
4. Write a JAVA program which contains a method fun() such that fun(x) returns x and fun(x,y) returns x2+ y2 (where x and y are integers).
5. Write a JAVA program which contains a method fun() such that fun(x) returns x and fun(x,y) returns x + y and fun(x,y,z) returns x*y*z. (where x, y and z are integers).
6. Write a set of overloaded methods min() that returns the smaller of two numbers passed to them as arguments. Make versions for int and float.
7. Write a set of overloaded methods power() that returns the Xn where n is integer and X maybe int and float.
8. Write a set of overloaded methods max() that returns the biggest of two numbers passed to them as arguments. Make versions for int and float.

WORKSPACE:

Exercise - 6

Aim: To study Inheritance and types of inheritance in java

Theory:

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.
In java extends keyword is used for inheritance.

Why use inheritance in java:

For Method Overriding (so runtime polymorphism can be achieved).
For Code Reusability.

Syntax of Java Inheritance:

```
class Subclass-name extends Superclass-name
{
        //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Example:-

```
class Employee
{
        float salary=40000;
}
class Programmer extends Employee
{
        int bonus=10000;
}
Class emp
{
        public static void main(String args[])
        {
                Programmer p=new Programmer();
                System.out.println("Programmer salary is:"+p.salary);
                System.out.println("Bonus of Programmer is:"+p.bonus);
        }
}
```

Output:

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

Method Overriding in Java:-

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

Method overriding is used to provide specific implementation of a method that is already provided by its super class.

Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

Method must have same name as in the parent class

Method must have same parameter as in the parent class.

Must be IS-A relationship (inheritance).

Example of method overriding:-

```
class Vehicle
{
	void run()
	{
		System.out.println("Vehicle is running");
	}
}
class Bike2 extends Vehicle
{
	void run()
	{
		System.out.println("Bike is running safely");
	}
}
class Mainmethodoverriding
{
	public static void main(String args[])
	{
		Bike2 obj = new Bike2();
		obj.run();
	}
}
```

Output:

Bike is running safely

"super" keyword in java:-

The super keyword in java is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of java super Keyword

super is used to refer immediate parent class instance variable.
super() is used to invoke immediate parent class constructor.
super is used to invoke immediate parent class method.
super is used to refer immediate parent class instance variable.

Problem without super keyword

```
class Vehicle
{
        int speed=50;
}
class Bike3 extends Vehicle
{
        int speed=100;
        void display()
        {
                System.out.println(speed);//will print speed of Bike
        }
}
Class Mainsuoerkey
{
        public static void main(String args[])
        {
                Bike3 b=new Bike3();
                b.display();
         }
}
```

Output:100

In the above example Vehicle and Bike both class have a common property speed. Instance variable of current class is refered by instance bydefault, but I have to refer parent class instance variable that is why we use super keyword to distinguish between parent class instance variable and current class instance variable.

Solution by super keyword

```
class Vehicle
{
        int speed=50;
}
class Bike4 extends Vehicle
```

```
{
	int speed=100;
	void display()
	{
		System.out.println(super.speed);//will print speed of Vehicle now
	}
}
Class Mainsuoerex1
{
	public static void main(String args[])
	{
		Bike4 b=new Bike4();
		b.display();
	 }
}
```
Output:50

super is used to invoke parent class constructor.
The super keyword can also be used to invoke the parent class constructor as given below:

Note: super() is added in each class constructor automatically by compiler.
As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement. If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.
Another example of super keyword where super() is provided by the compiler implicitly.

```
class Vehicle
{
	Vehicle()
	{
		System.out.println("Vehicle is created");
	}
}
class Bike6 extends Vehicle
{
	int speed;
	Bike6(int speed)
	{
		this.speed=speed;
		System.out.println(speed);
	}
	public static void main(String args[])
	{
		Bike6 b=new Bike6(10);
	}
}
```

Output:
Vehicle is created 10

super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
class Person
{
	void message()
	{
		System.out.println("Good Morning");
	}
}
class Student16 extends Person
{
	void message()
	{
		System.out.println("Welcome to java ");
	}
	void display()
	{
		message(); //will invoke current class message() method
		super.message(); //will invoke parent class message() method
	}
	public static void main(String args[])
	{
		Student16 s=new Student16();
		s.display();
	}
}
```

Output:
Welcome to java
Good morning

In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.
In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.
Program in case super is not required:

```
class Person
{
```

```
	void message()
	{
		System.out.println("welcome");
	}
}
class Student17 extends Person
{
	void display()
	{
		message();//will invoke parent class message()
	}
	public static void main(String args[])
	{
		Student17 s=new Student17();
		s.display();
	}
}
```

Output:

welcome

Exercise:-

1. Wap to find out volume of a box using simple inheritance ?
2. Wap to find out volume, cost and weight of a box using multilevel inheritance ?
3. Wap to find out volume, cost and weight of a box using multilevel inheritance and use super keyword at appropriate place?
4. Wap to use super keyword to access super class variable ?
5. Wap to use super keyword to access super class method ?
6. Wap to find out area and perimeter of rectangle and square using method overriding and abstract class ?
7. Wap to find out area and perimeter of rectangle and square using method overriding and abstract class and dynamic method dispatch?

WORKSPACE:

Exercise - 7

Aim: To study interface in java

Theory:

An interface in java is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
Java Interface also represents IS-A relationship. It cannot be instantiated just like abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.:
It is used to achieve fully abstraction.
By interface, we can support the functionality of multiple inheritance.
It can be used to achieve loose coupling.

Declaring Interfaces:

The interface keyword is used to declare an interface. Here is a simple example to declare an interface:
Example:
Let us look at an example that depicts encapsulation:

```
public interface NameOfInterface
{
        //Any number of final, static fields
        //Any number of abstract method declarations
}
```

Interfaces have the following properties:
An interface is implicitly abstract.
You do not need to use the abstract keyword when declaring an interface.
Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
Methods in an interface are implicitly public.
Example:

```
interface Animal
{
        public void eat();
        public void travel();
}
```

Implementing Interfaces:
When a class implements an interface, you can think of the class as signing a contract, agreeing to

perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
public class Mammal implements Animal
{
        public void eat()
        {
                System.out.println("Mammal eats");
        }
        public void travel()
        {
                System.out.println("Mammal travels");
        }
        public int noOfLegs()
        {
                return 0;
        }
        public static void main(String args[])
        {
                Mammal m =new Mammal();
                m.eat();
                m.travel();
        }
}
```

This would produce the following result:

```
Mammal eats
Mammal travels
```

Example:-

```
interface printable
{
        void print();
}
class A6 implements printable
{
        public void print()
        {
                System.out.println("Hello");
        }
        public static void main(String args[])
        {
                A6 obj = new A6();
                obj.print();
```

```
        }
}
```

Output:
Hello

Exercise:-

1. WAP to find out factorial of a number using interface ?
2. WAP to find out greatest among three numbers using interface?
3. WAP to find out gcd and lcm using interface ?
4. WAP to find out area and perimeter of rectangle using interface ?
5. WAP to find out area and perimeter of square using interface ?
6. WAP to find out area and perimeter of traingle using interface ?
7. WAP to find out area and perimeter of circle using interface ?
8. WAP to find out area and perimeter of rectangle ,square and triangle using interface and method overloading ?
9. WAP to find out area and perimeter of rectangle ,square and triangle using interface and method overriding ?

WORKSPACE

Exercise - 8

Aim: To study packages in JAVA

Theory:
A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc. Here, we will have the detailed learning of creating and using user-defined packages.
usage of classes, interfaces, enumerations and annotations easier, etc.

A Package can be defined as a grouping of related types(classes, interfaces, enumerations and annotations) providing access protection and name space management.

Some of the existing packages in Java are:

java.lang - bundles the fundamental classes.
java.io - classes for input , output functions are bundled in this package.

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.
Advantage of Java Package

Java package is used to categorize the classes and interfaces so that they can be easily maintained.
Java package provides access protection.
Java package removes naming collision.
Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating .

Creating a package:

When creating a package, you should choose a name for the package and put a package statement with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package.

The import Keyword:

If a class wants to use another class in the same package, the package name does not need to be used. Classes in the same package find each other without any special syntax.

Example:

Let us look at an example that creates a package called animals. It is common practice to use lowercased names of packages to avoid any conflicts with the names of classes, interfaces.

Put an interface in the package animals:

```
package animals;

interface Animal
{
        public void eat();
        public void travel();
}
```

Now, put an implementation in the same package animals:

```
import animals;

public class Mammal implements Animal
{
        public void eat()
        {
                System.out.println("Mammal eats");
        }
        public void travel()
        {
                System.out.println("Mammal travels");
        }
        public int noOfLegs()
        {
                return 0;
        }
        public static void main(String args[])
        {
                Mammal m =new Mammal();
                m.eat();
                m.travel();
        }
}
```

Now, you compile these two files and put them in a sub-directory called animals and try to run as follows:

C:> javac Animal.java
C:> javac Mammal.java
C:> java Mammal
Mammal eats
Mammal travels

Exercise:-

1. WAP to check wheather a number is palindrome or not using package ?
2. WAP to find out square root of a number without using sqrt or pow function and use package ?
3. WAP to check wheather a number is prime or not using package ?
4. WAP to find out greatest among 3 numbers using package ?
5. WAP to find out gcd and lcm using package ?

WORKSPACE

Exercise - 9

Aim: To study exception handling in JAVA

Theory:

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:
A user has entered invalid data. A file that needs to be opened cannot be found.
A network connection has been lost in the middle of communications or the JVM has run out of memory.
Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.
To understand how exception handling works in Java, you need to understand the three categories of exceptions:

Checked exceptions: A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
Runtime exceptions: A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
Errors: These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Catching Exceptions:

A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
//Protected code
}catch(ExceptionName e1)
{
//Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Example:

The following is an array is declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
import java.io.*;
public class ExcepTest
{
        public static void main(String args[])
        {
                try
                {
                        int a[] = {10,20,30};
                        System.out.println("Access element three :" + a[3]);
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                        System.out.println("Exception thrown :" + e);
                }
                System.out.println("Out of the block");
        }
}
```

This would produce the following result:

```
Exception thrown :
java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

Multiple catch Blocks:

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
//Protected code
}catch(ExceptionType1 e1)
{
//Catch block1
}catch(ExceptionType2 e2)
{
//Catch block2
}catch(ExceptionType3 e3)
{
//Catch block3
}
```

The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Example:
Here is code segment showing how to use multiple try/catch statements.

```
try
{
        file = new FileInputStream(fileName);
        x = (byte) file.read();
}
catch(IOException i)
{
        i.printStackTrace();
        return -1;
}
catch(FileNotFoundException f)
{
        f.printStackTrace();
        return -1;
}
```

The throws/throw Keywords:

If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the throw keyword. Try to understand the different in throws and throw keywords. The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
{
        public void deposit(double amount) throws RemoteException
        {
                // Method implementation throw new RemoteException();
        }
        //Remainder of class definition
}
```

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a RemoteException and an InsufficientFundsException:

```
import java.io.*;
public class className
{
        public void withdraw(double amount) throws RemoteException, InsufficientFundsException
        {
                // Method implementation
        }
        //Remainder of class definition
}
```

The finally Keyword

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
//Protected code
}catch(ExceptionType1 e1)
{
//Catch block1
}catch(ExceptionType2 e2)
{
//Catch block2
}catch(ExceptionType3 e3)
{
//Catch block3
}finally
{
//The finally block always executes.
}
```

Example:

```
public class ExcepTest
{
	public static void main(String args[])
	{
		int a[] = {10,20,30};
		try
		{
			System.out.println("Access element three :" + a[3]);
		}
		catch(ArrayIndexOutOfBoundsException e)
		{
			System.out.println("Exception thrown :" + e);
		}
		finally
		{
			a[0] = 6;
			System.out.println("First element value: " +a[0]);
			System.out.println("The finally statement is executed");
		}
	}
}
```

This would produce the following result:

Exception thrown :
java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed

Note the following:
A catch clause cannot exist without a try statement.
It is not compulsory to have finally clauses when ever a try/catch block is present.
The try block cannot be present without either catch clause or finally clause.
Any code cannot be present in between the try, catch, finally blocks.

Declaring your own Exception:

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

All exceptions must be a child of throwable.
If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
If you want to write a runtime exception, you need to extend the RuntimeException class.
We can define our own Exception class as below:

```
class MyException extends Exception{
}
```

You just need to extend the Exception class to create your own Exception class. These are considered to be checked exceptions. The following InsufficientFundsException class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

Example:

```
import java.io.*;
public class InsufficientFundsException extends Exception
{
        private double amount;
        public InsufficientFundsException(double amount)
        {
                this.amount = amount;
        }
        public double getAmount()
        {
                return amount;
        }
}
```

To demonstrate using our user-defined exception, the following CheckingAccount class contains a withdraw() method that throws an InsufficientFundsException.

```
// File Name CheckingAccount.java
```

```
import java.io.*;

public class CheckingAccount
{
	private double balance;
	private int number;
	public CheckingAccount(int number)
	{
		this.number = number;
	}
	public void deposit(double amount)
	{
		balance += amount;
	}
	public void withdraw(double amount) throws InsufficientFundsException
	{
		if(amount <= balance)
		{
			balance -= amount;
		}
		else
		{
			double needs = amount - balance;
			throw new InsufficientFundsException(needs);
		}
	}
	public double getBalance()
	{
		return balance;
	}
	public int getNumber()
	{
		return number;
	}
}
```

The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.

```
// File Name BankDemo.java
public class BankDemo
{
	public static void main(String [] args)
	{
		CheckingAccount c = new CheckingAccount(101);
		System.out.println("Depositing $500...");
		c.deposit(500.00);
		try
		{
			System.out.println("\nWithdrawing $100...");
			c.withdraw(100.00);
```

```
                System.out.println("\nWithdrawing $600...");
                c.withdraw(600.00);
            }catch(InsufficientFundsException e)
            {
                System.out.println("Sorry, but you are short $"+ e.getAmount());
                e.printStackTrace();
            }
        }
}
```

Compile all the above three files and run BankDemo, this would produce the following result:
Depositing $500...
Withdrawing $100...
Withdrawing $600...
Sorry, but you are short $200.0 InsufficientFundsException
at CheckingAccount.withdraw(CheckingAccount.java:lineno) at
BankDemo.main(BankDemo.java:lineno)

EXERCISE:-

1. Wap to create your own Exception to check whether a number is Prime or not, Palindrome or not, even or odd.
2. Wap to illustrate Arithemtic Exception,ArrayIndexOut Of BoundsEcxeption using nested Try Block with suitable example?
3. Wap to detect and resolve devide by zero error using Exception with suitable example?

WORKSPACE

Exercise - 10

Aim: To study threads and multithreading in java

Theory:

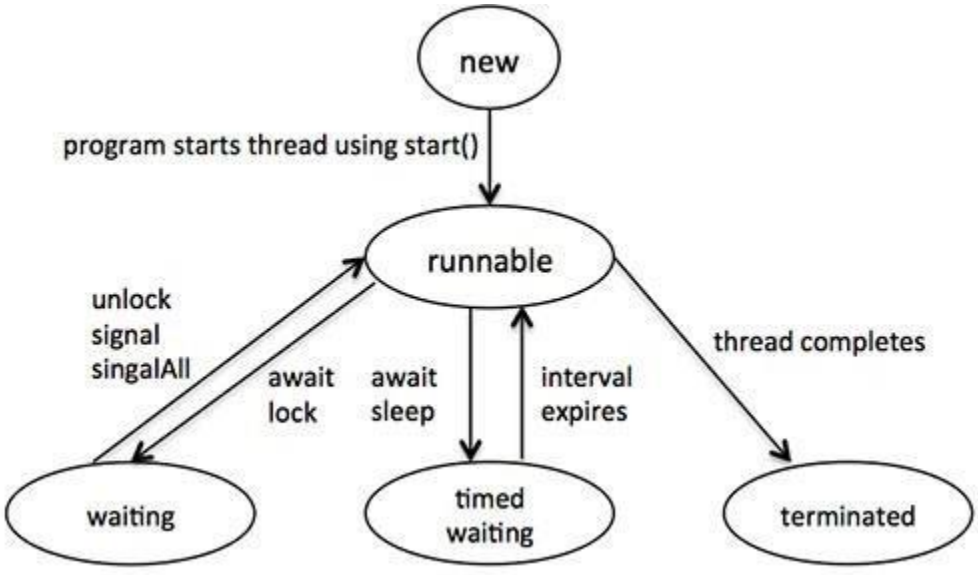
Java is a multithreaded programming language which means we can develop multithreaded program using Java. A multithreaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when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

By definition multitasking is when multiple processes share common processing resources such as a CPU. Multithreading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multithreading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Life Cycle of a Thread:

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.



Above-mentioned stages are explained here:

New: A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
Runnable: After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
Waiting: Sometimes, a thread transitions to the waiting state while the thread waits for another

thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
Timed waiting: A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
Terminated: A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Priorities:

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower- priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependentant.

Create Thread by Implementing Runnable Interface:

If your class is intended to be executed as a thread then you can achieve this by implementing Runnable interface. You will need to follow three basic steps:

Step 1:

As a first step you need to implement a run() method provided by Runnable interface. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:

public void run( )

Step 2:

At second step you will instantiate a Thread object using the following constructor:
Thread(Runnable threadObj, String threadName);
Where, threadObj is an instance of a class that implements the Runnable interface and threadName is the name given to the new thread.

Step 3

Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

void start( );

Example:
Here is an example that creates a new thread and starts it running:

```
class RunnableDemo implements Runnable
{
        private Thread t;
        private String threadName;
        RunnableDemo( String name)
        {
                threadName = name;
                System.out.println("Creating " + threadName );
        }
        public void run()
        {
                System.out.println("Running " + threadName );
                try {
                        for(int i = 4; i > 0; i--)
                        {
                                System.out.println("Thread: " + threadName + ", " + i);
                                // Let the thread sleep for a while.
                                 Thread.sleep(50);
                        }
                } catch (InterruptedException e) {
                        System.out.println("Thread " + threadName + " interrupted.");
                }
                System.out.println("Thread " + threadName + " exiting.");
        }

        public void start ()
        {
                System.out.println("Starting " + threadName );
                if (t == null)
                {
                        t = new Thread (this, threadName);
                        t.start ();
                }
        }
}
public  class TestThread
{
        public static void main(String args[])
        {
                RunnableDemo R1 = new RunnableDemo( "Thread-1");
                R1.start();
```

```
                RunnableDemo R2 = new RunnableDemo( "Thread-2");
                R2.start();
        }
}
```
This would produce the following result:
```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread:
Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Create Thread by Extending Thread Class:

The second way to create a thread is to create a new class that extends Thread class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1

You will need to override run( ) method available in Thread class. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:

```
public void run( )
```

Step 2

Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

```
void start( );
```

Example:
Here is the preceding program rewritten to extend Thread:
```
class ThreadDemo extends Thread
```

```
{
        private Thread t;
        private String threadName;
        ThreadDemo( String name)
        {
                threadName = name;
                System.out.println("Creating " + threadName );
        }
        public void run()
        {
                System.out.println("Running " + threadName );
                try {
                        for(int i = 4; i > 0; i--)
                        {
                                System.out.println("Thread: " + threadName + ", " + i);
                                // Let the thread sleep for a while.
                                Thread.sleep(50);
                        }
                } catch (InterruptedException e) {
                        System.out.println("Thread " + threadName + " interrupted.");
                }
                System.out.println("Thread " + threadName + " exiting.");
        }
        public void start ()
        {
                System.out.println("Starting " + threadName );
                if (t == null)
                {
                        t = new Thread (this, threadName);
                        t.start ();
                }
        }

}
public class TestThread
{
        public static void main(String args[])
        {
                ThreadDemo T1 = new ThreadDemo( "Thread-1");
                T1.start();
                ThreadDemo T2 = new ThreadDemo( "Thread-2");
                T2.start();
        }
}
```

This would produce the following result:

```
Creating Thread-1
Starting Thread-1
```

```
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread: Thread-1 exiting.
Thread: Thread-2 exiting.
```

Thread Methods:

Following is the list of important methods available in the Thread class.

SN	Methods with Description
1	public void start() Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	public void run() If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	public final void setName(String name) Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	public final void setPriority(int priority) Sets the priority of this Thread object. The possible values are between 1 and 10.
5	public final void setDaemon(boolean on) A parameter of true denotes this Thread as a daemon thread.
6	public final void join(long millisec) The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	public void interrupt() Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	public final boolean isAlive() Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.

SN	Methods with Description
1	public static void yield() Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.
2	public static void sleep(long millisec) Causes the currently running thread to block for at least the specified number of milliseconds.
3	public static boolean holdsLock(Object x) Returns true if the current thread holds the lock on the given Object.
4	public static Thread currentThread() Returns a reference to the currently running thread, which is the thread that invokes this method.
5	public static void dumpStack() Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Example:

Following is the main program which makes use of above defined classes:

```
public class ThreadClassDemo
{
        public static void main(String [] args)
        {
                Runnable hello = new DisplayMessage("Hello");
                Thread thread1 = new Thread(“hello”);
                thread1.setDaemon(true);
                thread1.setName("hello");
                System.out.println("Starting hello thread...");
                thread1.start();
                Runnable bye = new DisplayMessage("Goodbye");
                Thread thread2 = new Thread(“bye”);
                thread2.setPriority(Thread.MIN_PRIORITY);
                thread2.setDaemon(true);
                System.out.println("Starting goodbye thread...");
                thread2.start();
                System.out.println("Starting thread3...");
                Thread thread3 = new GuessANumber(27);
                thread3.start();
                try
                {
                        thread3.join();
                }catch(InterruptedException e)
                {
```

```
                    System.out.println("Thread interrupted.");
                }
                System.out.println("Starting thread4...");
                Thread thread4 = new GuessANumber(75);
                thread4.start();
                System.out.println("main() is ending...");
        }
}
```

This would produce the following result. You can try this example again and again and you would get different result every time.

Starting hello thread... Starting goodbye thread... Hello
Hello Hello Hello Hello Hello Goodbye Goodbye Goodbye Goodbye Goodbye
.....

Thread Synchronization:-

When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issue. For example if multiple threads try to write within a same file then they may corrupt the data because one of the threads can overrite data or while one thread is opening the same file at the same time another thread might be closing the same file.

So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called monitors. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

Java programming language provides a very handy way of creating threads and synchronizing their task by using synchronized blocks. You keep shared resources within this block. Following is the general form of the synchronized statement:

```
synchronized(objectidentifier){
// Access shared variables and other shared resources
}
```

Here, the objectidentifier is a reference to an object whose lock associates with the monitor that the synchronized statement represents. Now we are going to see two examples where we will print a counter using two different threads. When threads are not synchronized, they print counter value which is not in sequence, but when we print counter by putting inside synchronized() block, then it prints counter very much in sequence for both the threads.

Multithreading example without Synchronization:

Here is a simple example which may or may not print counter value in sequence and every time we run it, it produces different result based on CPU availability to a thread.

```
class PrintDemo
{
	public void printCount()
	{
		try{
			for(int i =5; i >0; i--)
			{
				System.out.println("Counter --- "+ i );
			}
		}catch(Exception e)
		{
			System.out.println("Thread interrupted.");
		}
	}
}
class ThreadDemo extends Thread
{
	private Thread  t;
	private String threadName;
	PrintDemo PD;
	ThreadDemo(String name,PrintDemo pd)
	{
		threadName = name;
		PD = pd;
	}
	public void run()
	{
		PD.printCount();
		System.out.println("Thread "+ threadName +" exiting.");
	}
	public void start ()
	{
		System.out.println("Starting "+ threadName );
		if(t ==null)
		{
			t =newThread(this, threadName);
			t.start ();
		}
	}
}
public class TestThread
{
	public static void main(String args[])
	{
		PrintDemo PD =new PrintDemo();
		ThreadDemo T1 =new ThreadDemo("Thread - 1 ", PD );
```

```
                ThreadDemo T2 =newThreadDemo("Thread - 2 ", PD );
                T1.start();
                T2.start();
        // wait for threads to end
                try{
                        T1.join();
                        T2.join();
                }catch(Exception e)
                {
                         System.out.println("Interrupted");
                }
        }
}
```

This produces different result every time you run this program:

```
StartingThread-1
StartingThread-2
Counter---5
Counter---4
Counter---3
Counter---5
Counter---2
Counter---1
Counter---4
Thread Thread-1 exiting.
Counter---3
Counter---2
Counter---1
Thread Thread-2 exiting.
```

Thread Control:-

Core Java provides a complete control over multithreaded program. You can develop a multithreaded program which can be suspended, resumed or stopped completely based on your requirements. There are various static methods which you can use on thread objects to control their behavior. Following table lists down those methods:

SN	Methods with Description
1	public void suspend() This method puts a thread in suspended state and can be resumed using resume() method.
2	public void stop() This method stops a thread completely.
3	public void resume() This method resumes a thread which was suspended using suspend() method.

4	public void wait() Causes the current thread to wait until another thread invokes the notify().
5	public void notify() Wakes up a single thread that is waiting on this object's monitor.

Exercise:-

1) Wap to perform following using Multithrading
    i) Addition of two nos.
    ii) Subtraction of two nos.
    iii) Multiplication of two nos.
    iv) Division of two nos.
    v) Modulus of two nos.
2) Wap to perform following using Multithrading
    i) Factorial of a number
    ii) Check wheaher a number is prime or not
    iii) Find gretest among three numbers
    iv) LCM
    v) GCD
3) Wap to perform following using Multithrading
    i) Sort 10 numbers in ascending order
    ii) Find Avg. of 10 numbers
    iii) Search a number in an array
4) Wap to detect and resolve deadlock using suitable example?
5) Wap to illustrate the need of synchronization using suitable example?

WORKSPACE

Exercise - 11

Aim: To study java AWT

Theory:

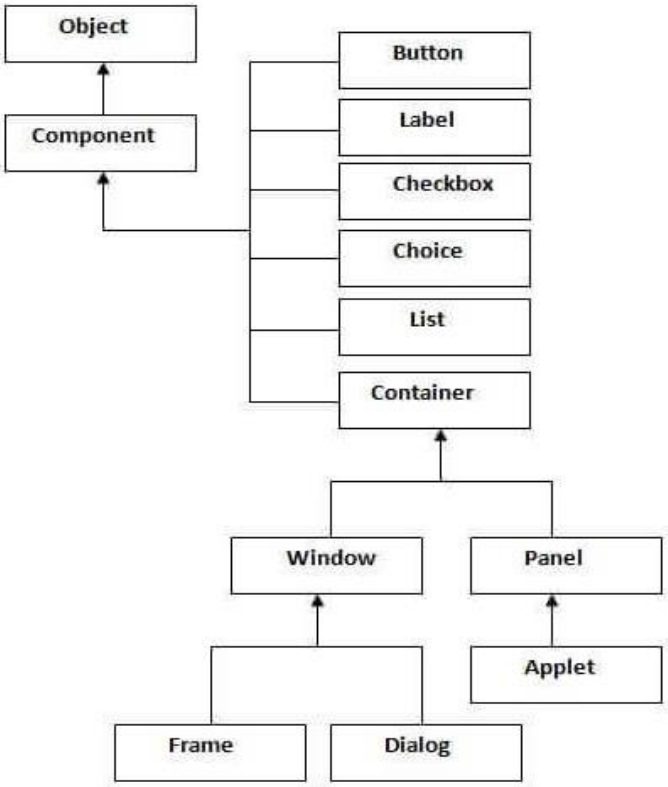
Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.

AWT is heavyweight i.e. its components are using the resources of OS.

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

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container:

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

Window:

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

Panel:

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

Frame:

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

Useful Methods of Component class:

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

Example:-

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

Event Handling:

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.
For registering the component with the Listener, many classes provide the registration methods. For example:

1) Button
    i) public void addActionListener(ActionListener a){}

2) MenuItem
    i) public void addActionListener(ActionListener a){}

3) TextField
    i) public void addActionListener(ActionListener a){}
    ii) public void addTextListener(TextListener a){}

4) TextArea
    i) public void addTextListener(TextListener a){}

5) Checkbox
    i) public void addItemListener(ItemListener a){}

6) Choice
    i) public void addItemListener(ItemListener a){}

7) List
    i) public void addActionListener(ActionListener a){}
    ii) public void addItemListener(ItemListener a){}

Example:-

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener
{
    TextField tf;
    AEvent()
    {
        //create components
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(100,120,80,30);

        //register listener
        b.addActionListener(this);//passing current instance
```

```
		//add components and set size, layout and visibility
		add(b);
		add(tf);
		setSize(300,300);
		setLayout(null);
		setVisible(true);
	}
	public void actionPerformed(ActionEvent e)
	{
		tf.setText("Welcome");
	}
	public static void main(String args[])
	{
		new AEvent();
	}
}
```

Exercise:-
Program to set the background color of panel using the color specified in the constants of the class.

WORKSPACE

Exercise - 12

Aim: To study applets in java

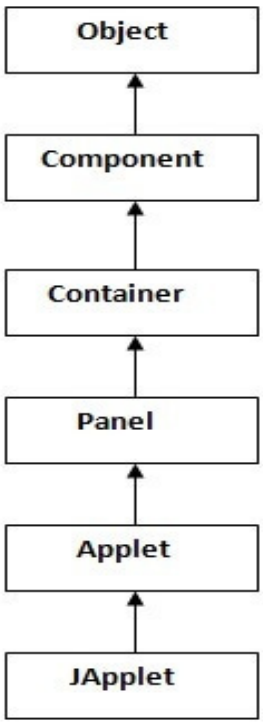
Theory:

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

Advantage of Applet

There are many advantages of applet. They are as follows:
It works at client side so less response time.
Secured.
It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Hierarchy of Applet:



Lifecycle of Java Applet:

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

Lifecycle methods for Applet:

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

java.applet.Applet class:

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.
public void init(): is used to initialized the Applet. It is invoked only once.
public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class:

The Component class provides 1 life cycle method of applet.

public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Graphics in Applet:

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

1) **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.

2) **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.

3) **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.

4) **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.

5) **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.

6) **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).

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

8) **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.

9) **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.

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

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

Example:

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

Applet File: save this file GraphicsDemo.java

```
import java.applet.Applet;
import java.awt.*;

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

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

        }
}
```

Html file: save this file GraphicsDemohtmlfile.html

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

```
</applet>
</body>
</html>
```

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

```
c:\>javac GraphicsDemo.java
c:\>appletviewer GraphicsDemohtmlfile.html
```

GUI Component in Applet:

GUI Components:

Label	Label noNameLabel = new Label();
	
	Label roadLabel = new Label("One Way");
	
	noNameLabel.setText("Two Way");
	
	add(noNameLabel);
	add(roadLabel);

Button	Button noNameButton = new Button();
	Button startButton = new Button("Start");
	
	String s = startButton.getLabel();
	
	Button stopButton = new Button();
	stopButton.setLabel("Stop");
	
	add(noNameButton);
	add(startButton);
	add(stopButton);

TextField	TextField f1 = new TextField();
	TextField f2 = new TextField();
	TextField f3 = new TextField(15);
	TextField f4 = new TextField("Hello");
	TextField f5 = new TextField("How are you",25);
	TextField f6 = new TextField(10);
	
	f2.setText("Good morning");
	
	// echo * whatever you input
	f6.setEchoChar('*');
	
	// passwd = whatever you input
	String passwd = f6.getText();

	add(f1); add(f2);
	add(f3); add(f4);
	add(f5); add(f6);

TextArea	TextArea t1 = new TextArea(2,10);
	TextArea t2 = new TextArea("Hello",4,20);
	
	add(t1);
	add(t2);

List	List years = new List();
	years.add("2007");
	years.add("2008");
	years.add("2009");
	years.add("2010");
	years.add("2011");
	years.add("2012");
	years.add("2013");
	years.add("2014");
	years.add("2015");
	
	add(years);

Scrollbar	// vertical scrollbar
	Scrollbar vscrollbar = new Scrollbar();
	
	// horizontal scrollbar
	Scrollbar hscrollbar =
	  new Scrollbar(Scrollbar.HORIZONTAL);
	
	hscrollbar.setUnitIncrement(20);
	hscrollbar.setMinimum(0);
	hscrollbar.setMaximum(100);
	
	add(vscrollbar);
	add(hscrollbar);

Choice	Choice threeChoices = new Choice();
	add(threeChoices);
	
	threeChoices.add("Choice 1");
	threeChoices.add("Choice 2");
	threeChoices.add("Choice 3");
	
	threeChoices.select("Choice 2");
	String s = threeChoices.getSelectedItem();

Checkbox	Checkbox checkbox1 = new Checkbox(); // no name Checkbox checkbox2 = new Checkbox("Any Time"); Checkbox checkbox3 = new Checkbox("Non-stop", true);

	add(checkbox1);
	add(checkbox2);
	add(checkbox3);

CheckboxGroup	CheckboxGroup TV = new CheckboxGroup();
	
	Checkbox c1 = new Checkbox("ABC 7", TV, false);
	Checkbox c2 = new Checkbox("FOX 11", TV, false);
	Checkbox c3 = new Checkbox("KRWG 22", TV, true);
	
	add(c1);
	add(c2);
	add(c3);

Panel	Panel p1 = new Panel();
	p1.add(new Button("Button in p1"));
	
	Panel p2 = new Panel();
	p2.add(new Button("Button in p2"));
	
	p1.add(p2);
	
	add(p1);

Components and their supported Events:

Component Type	Events supported
Adjustable	AdjustmentEvent
Applet	ContainerEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Button	ActionEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Canvas	FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Checkbox	ItemEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Choice	ItemEvent,FocusEvent,KeyEvent,MouseEbent,ComponentEvent
Component	FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Container	ContainerEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Frame	ContainerEvent,WindowEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Label	FocusEvent,KeyEvent,MouseEvent,ComponentEvent
List	ActionEvent,FocusEvent,KeyEvent,MouseEvent,ItemEvent
Menu	ActionEvent
MenuItem	ActionEvent

Panel	ContainerEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
PopupMenu	ActionEvent
Scrollbar	AdjustmentEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
TextArea	TextEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
TextComponent	TextEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
TextField	ActionEvent,TextEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent
Window	ContainerEvent,WindowEvent,FocusEvent,KeyEvent,MouseEvent,ComponentEvent

Example:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class ButtonApplet extends Applet implements ActionListener
{
        Button btn;
        Label lab;
        int counter;                        // by default assigned with 0
        public void init()
        {
                btn = new Button("Click And Count");
                lab = new Label("You click, I count");
                btn.addActionListener(this);
                add(btn);   add (lab);
         }
         public void actionPerformed(ActionEvent e)
         {
                 lab.setText("Your click count: " + counter++);
         }
}
```

Html File:

```
<html>
<body>
<applet
 code="ButtonApplet.class
" width="300"
height="200”>
</applet>
</body>
</html>
```

Assignment:

1. Create an applet for arithmetic operations.

WORKSPACE: