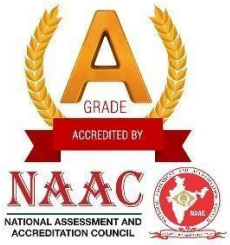
Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)



BHARATH INSTITUTE OF SCIENCE & TECHNOLOGY

173, Agaram Road, Selaiyur, Chennai-600073. Tamil Nadu, India.

SCHOOL OF COMPUTING

Department of Computer Science & Engineering

BACHELOR OF TECHNOLOGY

COURSE CODE: U20CSCJ05

Java Programming

LABORATORY MANUAL

INDEX

Ex. No : 1a) **Looping Statements - Odd or Even**
Date :

Aim:
To write a Java program to check whether the given number is odd or even.

Algorithm:
STEP 1: Start the Program.
STEP 2: Create the Class odd even
STEP 3: Read s values from keyboard using a Scanner class.
STEP 4: Check the (n % 2 == 0) value using if condition.
STEP 5: if the condition is satisfied number is even else the number is even.
STEP 6: Stop.

Source code:

```
import Java.util.Scanner;
public class Odd_Even
{
   public static void main(String[] args)
   {
int n;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the number you want to check:");
n = s.nextInt();
      if(n % 2 == 0)
      {
         System.out.println("The given number "+n+" is Even ");
      }
else
{
         System.out.println("The given number "+n+" is Odd ");
         }
   }
}
```

Output:

```
Enter the number you want to check:15 The
given number 15 is Odd
```

Result:
Thus, the above Java program to check whether the given number is odd or even was executed and the output was verified.

Ex. No: 1b) **Simple Calculator**
Date :

Aim:
To write a Java program to create a simple calculator using switch case statement.

Algorithm:
STEP 1: Start the Program.
STEP 2: Create the class calc
STEP 3: Read input values from keyboard using a Scanner class.
STEP 4: the various operators for calculator is done by using switch case statements various
STEP 5: Stop.

Program:

```
import Java.util.Scanner;
class calc {
  public static void main(String[] args) {

    char operator;
    Double number1, number2, result;

    // create an object of Scanner class
    Scanner input = new Scanner(System.in);

    // ask users to enter operator
    System.out.println("Choose an operator: +, -, *, or /");
operator = input.next().charAt(0);

    // ask users to enter numbers
    System.out.println("Enter first number");
number1 = input.nextDouble();

    System.out.println("Enter second number");
number2 = input.nextDouble();
    switch (operator) {
      // performs addition between numbers
case '+':
        result = number1 + number2;
        System.out.println(number1 + " + " + number2 + " = " + result);
break;

      // performs subtraction between numbers
case '-':
        result = number1 - number2;
        System.out.println(number1 + " - " + number2 + " = " + result);
break;

      // performs multiplication between numbers
case '*':
        result = number1 * number2;
```

```
        System.out.println(number1 + " * " + number2 + " = " + result);
    break;

       // performs division between numbers
    case '/':
        result = number1 / number2;
        System.out.println(number1 + " / " + number2 + " = " + result);
    break;

       default:
        System.out.println("Invalid operator!");
    break;
      }
      input.close();
    }
}
```

Output:

```
Choose an operator: +, -, *, or /
*
Enter first number
3
Enter second number
9
3.0 * 9.0 = 27.
```

Result:

Thus, the above Java program to create a simple calculator using switch case statement was executed and the output was verified.

Ex. No : 1c) **Sum of n Numbers using for Loop**
Date :

Aim:

To write a Java program to find the sum of n numbers using for loop.

Algorithm:

1. Start the program
2. Create a class with the name „ SumOfNNumbers ‟.
3. Read sc values from keyboard using a Scanner class.
4. Find the (n % i == 0) using dynamic initialization 5.

Check the (n % i == 0) value using if condition. STEP
1: Start the Program.
STEP 2: Create a class with the name SumOfNNumbers .
STEP 3: Read sc values from keyboard using a Scanner class.
STEP 4: Check for condition sum = sum +i using for loop
STEP 5: Stop.

Program:

```
import Java.util.Scanner;
public class SumOfNNumbers {
public static void main(String args[]){
    int sum = 0;
    System.out.print("Enter the number value:: ");
Scanner sc = new Scanner(System.in);
    int num = sc.nextInt();

    for (int i = 0; i<num; i++)
{
      sum = sum +i;
    }
    System.out.println("Sum of numbers : "+sum);
  }
}
```

Output

Enter the number value::

50

Sum of numbers : 1225

Result:

Thus, the above Java program to find the sum of n numbers using for loop was executed and the output was verified.

Ex. No : 1d) **Fibonacci Series**

Date :

Aim:

To write a Java program to find the Fibonacci series using While loop.

Algorithm:

STEP 1. Start the program

STEP 2. Create a class with the name Fibo.

STEP 3. Declare the int variables i, n, firstTerm, secondTerm.

STEP 4. Using while loop check whether i<=n

STEP 5.Stop

Program:

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

   int i = 1, n = 10, firstTerm = 0, secondTerm = 1;
   System.out.println("Fibonacci Series till " + n + " terms:");

   while (i <= n) {
     System.out.print(firstTerm + ", ");

     int nextTerm = firstTerm + secondTerm;
firstTerm = secondTerm;       secondTerm =
nextTerm;       i++;
   }
  }
}
```

Output

Fibonacci Series till 10 terms:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

Result:

Thus, the above Java program to find the Fibonacci series using while loop was executed and the output was verified.

Ex. No : 1e) **Sorting**
Date :

Aim:
To sort the array of numbers in ascending order using Java program.

Algorithm :
Setp1 : Start the program
Step2: Get the size of the array
Step3: Read the array elements
Step 4: Compare the successive elements of the array and arrange the elements in the ascending order
Step 5: Print the sorted array
Step 6: Stop the program.

Program :

```
import Java.util.Scanner;
public class ExArraySort {
  public static void main(String args[]) {
    // initialize the objects.
    int n, i, j, temp;
    int arr[] = new int[50];
    Scanner scan = new Scanner(System.in);

    // enter number of elements to enter.
    System.out.print("Enter number for the array elements : ");
    n = scan.nextInt();

    // enter elements.
    System.out.println("Enter " + n + " Numbers : ");
    for (i = 0; i < n; i++) {
      arr[i] = scan.nextInt();
    }

    // sorting array elements.
    System.out.print("Sorting array : \n");
    for (i = 0; i < (n - 1); i++) {
      for (j = 0; j < (n - i - 1); j++) {
        if (arr[j] > arr[j + 1]) {
          temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
        }
      }
    }

    System.out.print("Array Sorted Successfully..!!\n");

    // array in ascending order.
    System.out.print("Sorted List in Ascending Order : \n");
    for (i = 0; i < n; i++) {
```

```
      System.out.print(arr[i] + "  ");
    }
  }
}
```

Output:

```
Enter number for the array elements : 10
Enter 10 Numbers :
25
54
36
12
48
88
78
95
54
55
Sorting array :
Array Sorted Successfully..!!
Sorted List in Ascending Order :
12  25  36  48  54  54  55  78  88  95
```

Result :

Thus the Java program to sort the array elements is executed successfully and the output was verified.

Ex. No: 1f) **Matrix Addition**
Date :

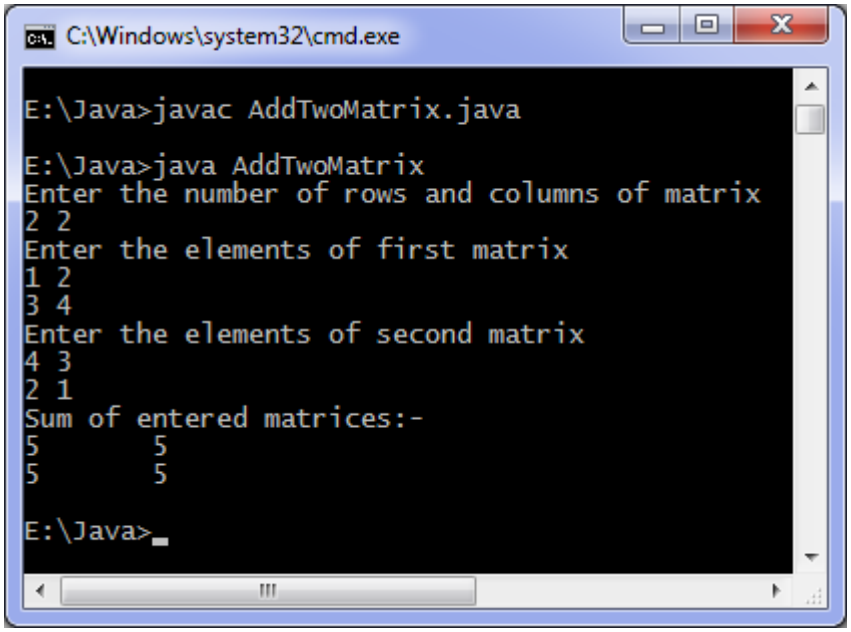
Aim :
To write a Java program to add the elements of two matrices.

Algorithm:
Step1 : Start the program
Step 2: Read the number of rows and columns of two matrices
Step 3: Read the elements of both the input matrices
Step 4: Add the elements of both the input matrices
Step 5: Print the resultant matrix
Step 6: Stop the program.

Program:

```
import Java.util.Scanner;
class AddTwoMatrix
{
  public static void main(String args[])
  {
    int m, n, c, d;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the number of rows and columns of matrix");
    m = in.nextInt();
    n = in.nextInt();
    int first[][] = new int[m][n];
    int second[][] = new int[m][n];
    int sum[][] = new int[m][n];
    System.out.println("Enter the elements of first matrix");
    for (c = 0; c < m; c++)
      for (d = 0; d < n; d++)
        first[c][d] = in.nextInt();
    System.out.println("Enter the elements of second matrix");
    for (c = 0 ; c < m; c++)
      for (d = 0 ; d < n; d++)
        second[c][d] = in.nextInt();
    for (c = 0; c < m; c++)
      for (d = 0; d < n; d++)
        sum[c][d] = first[c][d] + second[c][d];  //replace '+' with '-' to subtract matrices
    System.out.println("Sum of the matrices:");
    for (c = 0; c < m; c++)
    {
      for (d = 0; d < n; d++)
        System.out.print(sum[c][d] + "\t");
      System.out.println();
    }
  }
}
```

Output:



Result:

Thus the Java program to add two matrices has been executed successfully and the output was verified.

Ex. No : 2a) **Default Constructor**
Date :

Aim:

To write a Java program to display values using default constructor.

Algorithm:

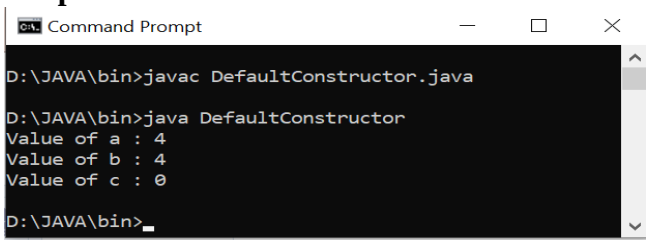
STEP 1: Start the Program.
STEP 2: Declare and Initialize the input variables.
STEP 3: Create the Constructor named Display.
STEP 4: Create various methods for Subclass.
STEP 5: In derived class extend the previous class.
STEP 6: In main class specify the values and create the object.
STEP 7: Stop.

Program:

```
class Display
 {
       int a=9; //initializer expression
       int b=4; //initializer expression
       int c;   //assigned default value
       Display()
       {

         a=4; //override default and initializer expression
       }
       void show()
       {
         System.out.println("Value of a : " +a);
         System.out.println("Value of b : " +b);
         System.out.println("Value of c : " +c);
       }
 }
   class DefaultConstructor
 {
     public static void main(String[] args)
     {
       Display data=new Display();
       data.show();
     }
 }
```

Output:



Result:

Thus, the above program to display values using default constructor has been executed and the output was verified.

Ex. No : 2b) **Parameterized Constructor**
Date :

Aim:

To write a Java program to display student details using parameterized constructor.

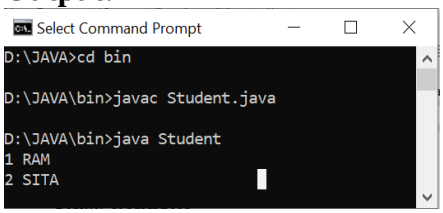
Algorithm:

STEP 1: Start the Program.
STEP 2: Declare and Initialize the input variables.
STEP 3: Create the Constructor named student.
STEP 4: Create various methods for Subclass.
STEP 5: In derived class extend the previous class.
STEP 6: In main class specify the values and create the object.
STEP 7: Stop.

Program:

```
class Student
{
int roll_no;
String stu_name;
Student(int i , String n)              // Parameterized constructor
{
roll_no = i;                    // Instance Variable
stu_name = n;                     // Instance Variable
}
void display()
{
System.out.println(roll_no+" "+stu_name);
}
public static void main(String args[])
{
Student s1 = new Student(1,"RAM");
Student s2 = new Student(2,"SITA");
s1.display();
s2.display();
}
}
```

Output:



Result:

Thus, the above program to display student details using parameterized constructor has been executed and the output was verified.

Ex. No 2c) **Copy Constructor**
Date :

Aim:

To write a Java program to find out the area of rectangle using copy constructor.

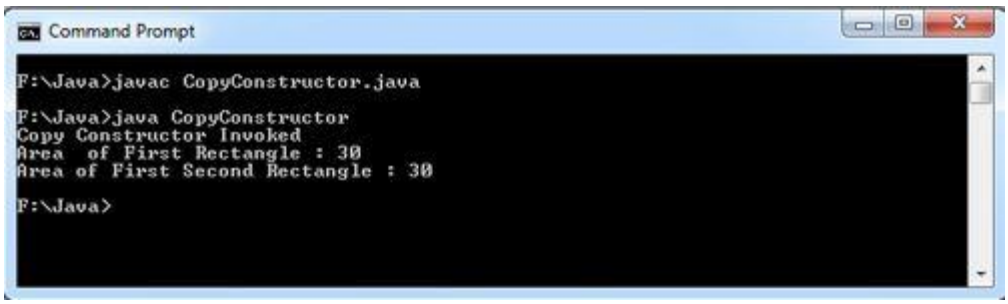
Algorithm:

STEP 1: Start the Program.
STEP 2: Declare and Initialize the input variables.
STEP 3: Create the Constructor named Rectangle.
STEP 4: Create various methods for Subclass.
STEP 5: In derived class extend the previous class.
STEP 6: In main class specify the values and create the object.
STEP 7: Stop.

Program:

```
class Rectangle
 {
       int length;
       int breadth;
       //constructor to initialize length and bredth of rectang of rectangle
       Rectangle(int l, int b)
       {
          length = l;
          breadth= b;
       }
       //copy constructor
       Rectangle(Rectangle obj)
       {
         System.out.println("Copy Constructor Invoked");
         length = obj.length;
         breadth= obj.breadth;
       }
      //method to calcuate area of rectangle
      int area()
      {
        return (length * breadth);
      }
 }
      //class to create Rectangle object and calculate area
      class CopyConstructor
 {
        public static void main(String[] args)
        {
          Rectangle firstRect = new Rectangle(5,6);
          Rectangle secondRect= new Rectangle(firstRect);
          System.out.println("Area  of First Rectangle : "+ firstRect.area());
          System .out.println("Area of First Second Rectangle : "+ secondRect.area());
        }
 }
```

Output:



Result:

Thus the copy constructor program has been executed and the output was verified successfully.

Ex. No 2d) **Method Overloading**
Date:

Aim:

To write a program in Java to implement method overloading

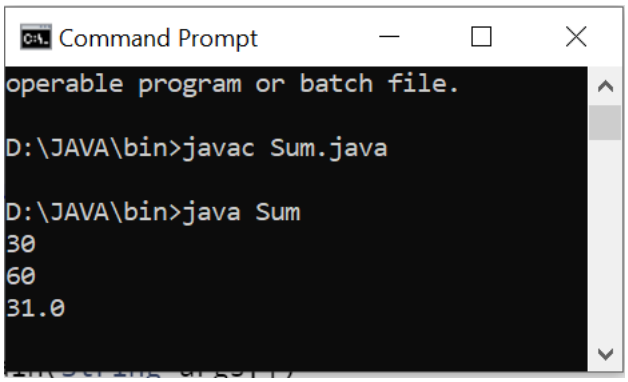
Algorithm:

STEP 1: Start the Program
STEP 2: Initialize the File Pointer
STEP 3: Create the class Sum
STEP 4: Overload Sum with two parameters
STEP 5: Create int sum
STEP 6: Create another overloaded sum with two double parameters
STEP 7: In main function create the object and call the methods
STEP 8: Print the data in the file

Program:

```
// Java program to demonstrate working of method
// overloading in Java.
public class Sum {
// Overloaded sum(). This sum takes two int parameters
public int sum(int x, int y)
{
return (x + y);
}
// Overloaded sum(). This sum takes three int parameters
public int sum(int x, int y, int z)
{
return (x + y + z);
}
// Overloaded sum(). This sum takes two double parameters
public double sum(double x, double y)
{
return (x + y);
}
public static void main(String args[])
{
Sum s = new Sum();
System.out.println(s.sum(10, 20));
System.out.println(s.sum(10, 20, 30));
System.out.println(s.sum(10.5, 20.5));
}
}
```

Output:



Result:

Thus, the Java program to implement method overloading was executed and the output was verified successfully.

Ex. No. 3a) **Parameter Passing – Call by Value**
Date :

Aim:
To write a Java program for implementing call by value technique.

Algorithm:
Step1: Start the program
Step2: Assign the values for the variables a & b
Step3: Pass the values to the method
Step4: Inside the method increment the values of a & b
Step5: Print the values of a & b
Step6: Stop the program

Program:

```
// Callee
class CallByValue {

   // Function to change the value
   // of the parameters
   public static void example(int x, int y)
   {
      x++;
      y++;
   }
}

// Caller
public class Main {
   public static void main(String[] args)
   {

      int a = 10;
      int b = 20;

      // Instance of class is created
      CallByValue object = new CallByValue();

      System.out.println("Value of a: " + a
                  + " & b: " + b);

      // Passing variables in the class function
      object.example(a, b);

      // Displaying values after
      // calling the function
      System.out.println("Value of a: "
                  + a + " & b: " + b);
   }
```

Output:

Value of a: 10 & b: 20

Value of a: 10 & b: 20

Result:

Thus the Java program to implement the call by value technique has been executed and the output was verified.

Ex. No : 3b) **Call by Reference**
Date :

Aim :

To write a Java program for implementing the call by reference technique.

Algorithm:

Step1: Start the program
Step2: Assign the values for the variables a & b
Step3: Pass the reference of the variables a & b
Step4: Add the values 10 and 20 to the variables a & b respectively.
Step5: Print the values of a & b
Step6: Stop the program.

Program:

```
//Java program to illustrate
// Call by Reference

// Callee
class CallByReference {

    int a, b;

    // Function to assign the value
    // to the class variables
    CallByReference(int x, int y)
    {
        a = x;
        b = y;
    }

    // Changing the values of class variables
    void ChangeValue(CallByReference obj)
    {
        obj.a += 10;
        obj.b += 20;
    }
}

// Caller
public class callbyref{

    public static void main(String[] args)
    {

        // Instance of class is created
        // and value is assigned using constructor
```

```
        CallByReference object
            = new CallByReference(10, 20);

        System.out.println("Value of a: " + object.a
                        + " & b: " + object.b);

        // Changing values in class function
        object.ChangeValue(object);

        // Displaying values
        // after calling the function
        System.out.println("Value of a: " + object.a
                        + " & b: " + object.b);
    }
}
```

Output:

Value of a: 10 & b: 20

Value of a: 20 & b: 40

Result:

Thus the Java program to implement the technique called call by reference has been executed successfully and the output was verified.

Ex. No: 3c) **Recursion**
Date :

Aim:
To write a Java program to find the factorial of a number recursively.

Algorithm:
Step1: Start the program
Step2: If the number is 1 then return the factorial as 1
Step3: Otherwise calculate the factorial of the number recursively using the formula
Result=fact(n-1)*n
Step4: Print the factorial of the number
Step5: Stop the program.

Program:

```
class GEG{
int fact(int n){
int result;

  if(n==1) return 1;
  result =fact(n-1) *n;
  return result;
}
}

class Recursion {

    public static void main (String[] args) {
      GEG f= new GEG ();

      System.out.println("Factorial of 3 is " + f.fact(3));
      System.out.println("Factorial of 4 is " + f.fact(4));
      System.out.println("Factorial of 5 is " + f.fact(5));
    }
}
```

Output:
Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120

Result:

Thus the program to find the factorial of the number recursively has been executed successfully and the output was verified.

Ex. No : 4 **String Handling**

Date :

Aim :

To write a Java program to perform the various operations on string.

Algorithm:

Step1: Start the program
Step2: Read two strings
Step3: Find out the length, lowercase, uppercase, substring of the strings using built in functions.
Step4: Print the output
Step5: Stop the program

Program:

```
class Main{
  public static void main(String []args)
  {
    String s1="Adithya";
    String s2="Adithya";
    String s3="Adi";
    boolean x=s1.equals(s2);
    System.out.println("Compare s1 and s2:"+x);
    System.out.println("Character at given position is:"+s1.charAt(5));
    System.out.println(s1.concat(" the author"));
    System.out.println(s1.length());
    System.out.println(s1.toLowerCase());
    System.out.println(s1.toUpperCase());
    System.out.println(s1.indexOf('a'));
    System.out.println(s1.substring(0,4));
    System.out.println(s1.substring(4));
  }
}
```

Output:

```
Compare s1 and s2 : true
Character at given position is : y
Adithya the author
7
adithya
ADITHYA
6
adit
hya
```

Result:

Thus the program to perform various operations on strings has been executed successfully and the output was verified.

Ex. No : 5a) **Single Inheritance**
Date :

Aim :

To write a Java program to implement the single inheritance.

Algorithm:

Step1: Start the program
Step2: Write parent class
Step3: Write the child class
Step4: Inherit parent class to child class
Step5: Print the output
Step 6: Stop the program

Program:

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

```
barking...
eating...
```

Result:

Thus the program to implement single inheritance has been executed successfully and the output was verified.

Ex. No : 5b) **Multilevel Inheritance**
Date :

Aim:

To write a Java program to implement the multilevel inheritance.

Algorithm:

Step1: Start the program
Step2: Write three classes namely Animal, Dog and babyDog
Step3: Inherit Animal class to Dog class
Step4: Inherit Dog class to BabyDog class
Step5: Print the output
Step6: Stop the program.

Program:

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

Result:

Thus the program to implement multilevel inheritance has been executed successfully and the output was verified.

Ex. No : 5c) **Hierarchical Inheritance**
Date :

Aim:

To write a Java program to implement the hierarchical inheritance.

Algorithm:

Step1: Start the program
Step2: Write three classes namely Animal, Dog and Cat
Step3: Inherit Animal class to Dog class
Step4: Inherit Animal class to Cat class.
Step5: Print the output
Step6: Stop the program

Program:

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Result:

Thus the program to implement hierarchical inheritance has been executed successfully and the output was verified.

Ex. No : 5d) **Hybrid Inheritance**
Date :

Aim:

To write a Java program to implement hybrid inheritance.

Algorithm:

Step1: Start the program
Step2: Write a class called GrandFather
Step3: Inherit the class GrandFather to the class Father by the concept of single inheritance
Step4: Inherit the class Father to the classes Son and Daughter based on the concept of hierarchical inheritance
Step5: Print the output
Step6: Stop the program.

Program:

```
class GrandFather

{

public void showG()

{

System.out.println("He is grandfather.");

}

}

//inherits GrandFather properties

class Father extends GrandFather

{

public void showF()

{

System.out.println("He is father.");

}

}

//inherits Father properties

class Son extends Father

{
```

```
public void showS()
{
System.out.println("He is son.");
}
}
//inherits Father properties
public class Daughter extends Father
{
public void showD()
{
System.out.println("She is daughter.");
}
public static void main(String args[])
{
//Daughter obj = new Daughter();
//obj.show();
Son obj = new Son();
obj.showS();  // Accessing Son class method
obj.showF();  // Accessing Father class method
obj.showG();  // Accessing GrandFather class method
Daughter obj2 = new Daughter();
obj2.showD();  // Accessing Daughter class method
obj2.showF();    // Accessing Father class method
obj2.showG();   // Accessing GrandFather class method
}
}
```

Output:

He is son.

He is father.

He is grandfather.

She is daughter.

He is father.

He is grandfather.

Result:

Thus the program to implement hybrid inheritance has been executed successfully and the output was verified.

Ex. No : 5e) **Method Overriding**
Date :

Aim:

To write a Java program to implement method overriding.

Algorithm:

Step1: Start the program
Step2: Write parent class with a method
Step3: Write the child class with a method which is same as that of parent class method
Step4: Inherit parent class to child class
Step5: Call the method from main()
Step6: The child class method will override the parent class method
Step7: Print the output
Step8: Stop the program.

Program:

```
class ParentClass{
  //Parent class constructor
ParentClass(){
System.out.println("Constructor of Parent");
  }
  void disp(){
System.out.println("Parent Method");
  }
}
class JavaExample extends ParentClass{
  JavaExample(){
System.out.println("Constructor of Child");
  }
  void disp(){
System.out.println("Child Method");
  }
  public static void main(String args[]){
//Creating the object of child class
JavaExample obj = new JavaExample();
obj.disp();
  }
}
```

Output :

```
Constructor of Child
Child Method
```

Result:

Thus the program for method overriding has been executed successfully and the output was verified.

Ex. No : 6a) **Exception Handling - Arithmetic Exception**

Date:

Aim :

To write a program in Java to generate arithmetic exception and display the message using try and catch blocks.

Algorithm:

Step 1: Start the program

Step 2: Create a class ArithmeticException_Demo and write the main method

Step 3: Use a try block where the exception is created.

Step 4: Use a catch block and print the error message.

Step 5: Stop the program

Program :

```
public class ArithmeticException_Demo
{
   public static void main(String args[])
   {
      try {
         int a = 30, b = 0;
         int c = a/b;
         System.out.println ("Result = " + c);
      }
      catch (ArithmeticException e) {
         System.out.println ("Can't divide a number by 0");
      }
   }
}
```

Output :

Can't divide a number by 0

Result:

Thus the program to generate arithmetic exception using try and catch blocks has been executed successfully and the output was verified.

Ex. No : 6b) **Exception Handling - ArrayIndexOutOfBounds**

Date:

Aim :

To write a program in Java to generate ArrayIndexOutOfBounds Exception and display the message using try and catch blocks.

Algorithm:

Step 1: Start the program

Step 2: Create a class ArrayIndexOutOfBound_Demo and write the main method

Step 3: Use a try block where the exception is created.

Step 4: Use a catch block and print the error message.

Step 5: Stop the program

Program :

```
public class ArrayIndexOutOfBound_Demo
{
   public static void main(String args[])
   {
      try{
         int a[] = new int[5];
         a[6] = 9; // accessing 7th element in an array of
                  // size 5
      }
      catch(ArrayIndexOutOfBoundsException e){
         System.out.println ("Array Index is Out Of Bounds");
      }
   }
}
```

Output:

Array Index is Out Of Bounds

Result:

Thus the program to generate ArrayIndexOutOfBounds exception using try and catch block has been executed successfully and the output was verified.

Ex. No : 6c) **Exception Handling – finally Block**
Date :

Aim :

To write a program in Java to generate Exception and display the message using try and finally blocks.

Algorithm:

Step 1: Start the program

Step 2: Create a class and write the main method

Step 3: Use a try block where the exception is created.

Step 4: Use a finally block and print the error message.

Step 5: Stop the program

Program :

```
public class Main {

public static void main (String[] args) {

 try{
          System.out.println(4/0);
       }
   finally
       {
          System.out.println("finally executed");
       }

                    System.out.println("end");
}
}
```

Output :

```
finally executed

Exception in thread "main" Java.lang.ArithmeticException: / by zero
at Main.main(Main.Java:6)
```

Result:

Thus the program to generate exception and display the message using try and finally blocks has been executed successfully and the output was verified.

Ex.No : 6d) **Exception Handling - throw Keyword**

Date :

Aim :

To write a program in Java to generate Exception explicitly using try and throw.

Algorithm:

Step 1: Start the program

Step 2: Create a class and write the main method

Step 3: Use a try block where the exception is created explicitly using throw keyword.

Step 4: Use a catch block and print the message.

Step 5: Stop the program

Program :

```
public class ExceptionDemo
{
static void canVote(int age)
{
      if (age<18)
         try {
            throw new Exception();
            }
      catch (Exception e)
        {System.out.println ("you are not an adult!");}
      else
        System.out.println ("you can vote!");
}
public static void main (String[] args) {
      canVote(20);
      canVote(10);
}
}
```

Output:

```
you can vote!
you are not an adult!
```

Result :

Thus the program to generate Exception explicitly using throw has been written and executed successfully and the output was verified.

Ex. No :6e) **Exception Handling – throws Keyword**

Date :

Aim :

To write a program in Java to generate Exception explicitly using try and throws keyword.

Algorithm:

Step 1: Start the program

Step 2: Create a class and write the main method

Step 3: Use a try block where the exception is created explicitly using throws keyword.

Step 4: Use a catch block and print the message.

Step 5: Stop the program

Program:

```
public class ExceptionDemo {
static void func(int a) throws Exception{
        System.out.println(10/a);
}
public static void main (String[] args) {
      try{
         func(10);
         func(0);
      }
       catch (Exception e){
        System.out.println ("can't divide by zero");
      }

}
}
```

Output :

```
1
can't divide by zero
```

Result:

Thus the program to generate exception explicitly using throws has been written and executed successfully and the output was verified.

Ex. No : 7 **Simple Calculator using Grid Layout**
Date :

Aim:

To write a Java program to create a simple calculator using grid layout.

Algorithm

Step1: Start the program
Step2: Initialize JButton[]
Step3: Set the size of the frame
Step4: Set the rows and columns of the grid
Step5: Add a text field with a specified text to the frame
Step6: Add buttons to the frame
Step7: Implement ActionListener to perform the action invoked on the buttons
Step8: Stop the program

Program:

```
import Java.awt.*;
import Java.awt.event.*;
import Javax.swing.*;

class BuildCalculator extends JFrame implements ActionListener{
JFrame actualWindow;
JPanel resultPanel, buttonPanel, infoPanel;

JTextField resultTxt;
JButton btn_digits[] = new JButton[10];
JButton btn_plus, btn_minus, btn_mul, btn_div, btn_equal, btn_dot, btn_clear;

char eventFrom;

JLabel expression, appTitle, siteTitle ;

double oparand_1 = 0, operand_2 = 0;
String operator = "=";

BuildCalculator() {
    Font txtFont = new Font("SansSerif", Font.BOLD, 20);
    Font titleFont = new Font("SansSerif", Font.BOLD, 30);
    Font expressionFont = new Font("SansSerif", Font.BOLD, 15);

    actualWindow = new JFrame("Calculator");
    resultPanel = new JPanel();
    buttonPanel = new JPanel();
    infoPanel = new JPanel();

    actualWindow.setLayout(new GridLayout(3, 1));
```

```
buttonPanel.setLayout(new GridLayout(4, 4));
infoPanel.setLayout(new GridLayout(3, 1));
actualWindow.setResizable(false);


appTitle = new JLabel("My Calculator");
appTitle.setFont(titleFont);
expression = new JLabel("Expression shown here");
expression.setFont(expressionFont);
siteTitle = new JLabel("www.btechsmartclass.com");
siteTitle.setFont(expressionFont);
siteTitle.setHorizontalAlignment(SwingConstants.CENTER);
siteTitle.setForeground(Color.BLUE);

resultTxt = new JTextField(15);
resultTxt.setBorder(null);
resultTxt.setPreferredSize(new Dimension(15, 50));
resultTxt.setFont(txtFont);
resultTxt.setHorizontalAlignment(SwingConstants.RIGHT);

for(int i = 0; i <10; i++) {
        btn_digits[i] = new JButton(""+i);
        btn_digits[i].addActionListener(this);
}
btn_plus = new JButton("+");
btn_plus.addActionListener(this);
btn_minus = new JButton("-");
btn_minus.addActionListener(this);
btn_mul = new JButton("*");
btn_mul.addActionListener(this);
btn_div = new JButton("/");
btn_div.addActionListener(this);
btn_dot = new JButton(".");
btn_dot.addActionListener(this);
btn_equal = new JButton("=");
btn_equal.addActionListener(this);
btn_clear = new JButton("Clear");
btn_clear.addActionListener(this);

resultPanel.add(appTitle);
resultPanel.add(resultTxt);
resultPanel.add(expression);
for(int i = 0; i < 10; i++) {
        buttonPanel.add(btn_digits[i]);
}
buttonPanel.add(btn_plus);
buttonPanel.add(btn_minus);
buttonPanel.add(btn_mul);
buttonPanel.add(btn_div);
buttonPanel.add(btn_dot);
buttonPanel.add(btn_equal);
```

```
    infoPanel.add(btn_clear);
    infoPanel.add(siteTitle);

    actualWindow.add(resultPanel);
    actualWindow.add(buttonPanel);
    actualWindow.add(infoPanel);



    actualWindow.setSize(300, 500);
    actualWindow.setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    eventFrom = e.getActionCommand().charAt(0);
    String buildNumber;
    if(Character.isDigit(eventFrom)) {
            buildNumber = resultTxt.getText() + eventFrom;
            resultTxt.setText(buildNumber);
    } else if(e.getActionCommand() == ".") {
            buildNumber = resultTxt.getText() + eventFrom;
            resultTxt.setText(buildNumber);
    }
    else if(eventFrom != '='){
            oparand_1 = Double.parseDouble(resultTxt.getText());
            operator = e.getActionCommand();
            expression.setText(oparand_1 + " " + operator);
            resultTxt.setText("");
    } else if(e.getActionCommand() == "Clear") {
            resultTxt.setText("");
    }
    else {
            operand_2 = Double.parseDouble(resultTxt.getText());

            expression.setText(expression.getText() + " " + operand_2);
            switch(operator) {
                    case "+":       resultTxt.setText(""+(oparand_1 + operand_2)); break;
                    case "-":       resultTxt.setText(""+(oparand_1 - operand_2)); break;
                    case "*":       resultTxt.setText(""+(oparand_1 * operand_2)); break;
                    case "/":       try {
                                            if(operand_2 == 0)
                                                    throw new ArithmeticException();
                                            resultTxt.setText(""+(oparand_1 /
operand_2)); break;
                                    } catch(ArithmeticException ae) {

    JOptionPane.showMessageDialog(actualWindow, "Divisor can not be ZERO");
                                    }
            }
    }
```

```
}
}

public class Calculator {
public static void main(String[] args) {
    new BuildCalculator();
}
}
```

Output:

Result:

Thus the program to create a simple calculator has been written and executed successfully and the output was verified.

Ex. No : 8 **Producer Consumer Problem using Inter Thread**

Date : **Communication**

Aim:

To write a Java program to implement the concept of producer - consumer problem using inter thread communication.

Algorithm:

Step1: Start the program
Step2: Create the producer thread
Step3: Create the consumer thread
Step4: Make the producer to produce a data in the buffer
Step5: Make the consumer to consume the data from the buffer
Step6: Print the output
Step 7: Stop the program

Program:

```
import Java.util.LinkedList;
public class Threadexample {
public static void main(String[] args)
    throws InterruptedException
{
    // Object of a class that has both produce() and consume() methods
    final PC pc = new PC();

    // Create producer thread
    Thread t1 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                    try {
                            pc.produce();
                    }
                    catch (InterruptedException e) {
                            e.printStackTrace();
                    }
            }
    });

    // Create consumer thread
    Thread t2 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                    try {
                            pc.consume();
                    }
```

```
                catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
    });

    // Start both threads
    t1.start();
    t2.start();

    // t1 finishes before t2
    t1.join();
    t2.join();
}

// This class has a list, producer (adds items to list and consumer (removes items).
public static class PC {
    // Create a list shared by producer and consumer  Size of list is 4.
    LinkedList<Integer> list = new LinkedList<>();
    int capacity = 4;

    // Function called by producer thread
    public void produce() throws InterruptedException
    {
        int value = 0;
        while (true) {
            synchronized (this)
            {
                // producer thread waits while list  is full
                while (list.size() == capacity)
                    wait();

                System.out.println("Producer produced-"
                                                + value);

                // to insert the jobs in the list
                list.add(value++);

                    // notifies the consumer thread that  nowit can start
                    //consuming
                notify();

                // makes the working of program easier
                // to understand
                Thread.sleep(1000);
            }
        }
    }

    // Function called by consumer thread
    public void consume() throws InterruptedException
    {
        while (true) {
```

```
                synchronized (this)
                {
                        // consumer thread waits while list
                        // is empty
                        while (list.size() == 0)
                                wait();

                        // to retrieve the first job in the list
                        int val = list.removeFirst();

                        System.out.println("Consumer consumed-"
                                                        + val);

                        // Wake up producer thread
                        notify();

                        // and sleep
                        Thread.sleep(1000);
                }
        }
    }
}
}
```

Output :

```
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
```

Result :

Thus the program to implement the concept of producer - consumer problem using inter thread communication has been executed successfully and the output was verified.

Ex. No : 9 **Applet Program**
Date :

Aim:

To write a program for finding the factorial of a number using Applet.

Algorithm:

Step1: Start the program.
Step2: Create a class with the name “fact” extends Applet which implements ActionListener.
Step3: Declare the Lable Textfield, Button variables.
Step4: Call init method.
Step5: Call the setLayout method,setLayout(g).
Step6: Create the HTML file for applet tag.
Step7: Stop the program

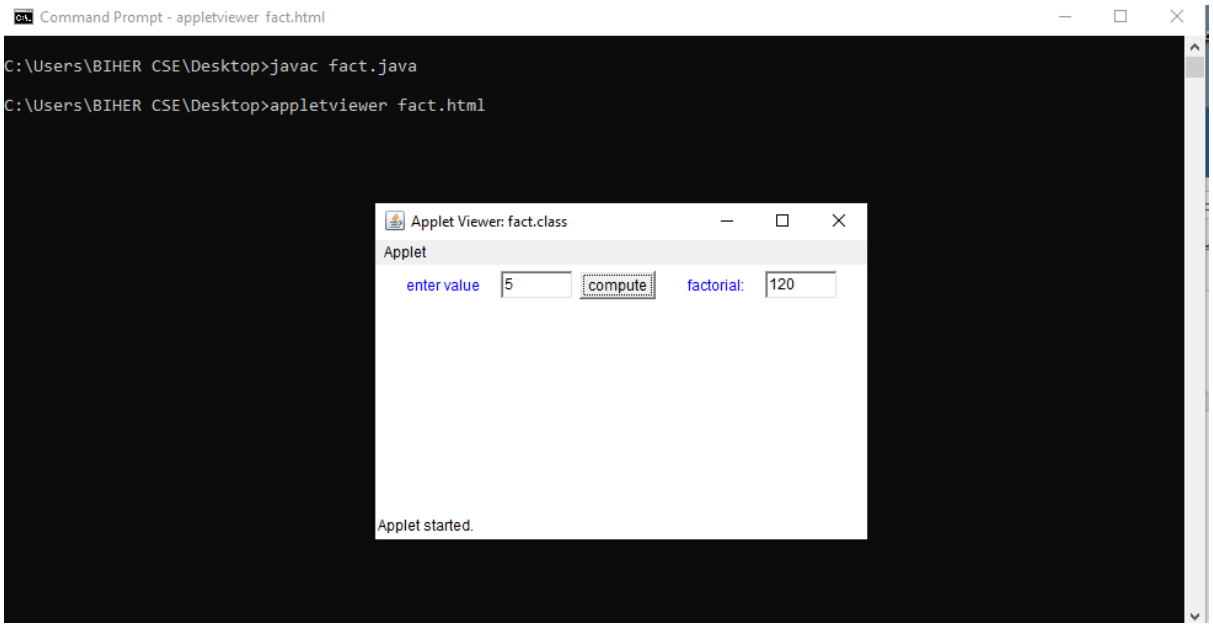
Program:

```
import Java.applet.Applet;
import Java.awt.*;
import Java.awt.event.*;
/* <applet code=”fact.class” width=300 height=300>
</applet>*/

public class fact extends Applet implements ActionListener
{
  Label i1,i2,i3;
  TextField tf1,tf2;
  Button b1;
  public void init()
  {
     setSize(400,200);
     FlowLayout g=new FlowLayout();
     setLayout(g);
     i1 = new Label("enter value");
     i1.setForeground(Color.BLUE);
     add(i1);
     tf1 = new TextField(5);
     tf1.setText("0");
     add(tf1);
     b1 = new Button("compute");
     b1.addActionListener(this);
     add(b1);
     i3 = new Label();
     add(i3);
     i2 = new Label("factorial:");
     i2.setForeground(Color.BLUE);
     add(i2);
     tf2 = new TextField(5);
     add(tf2);
}
public void actionPerformed(ActionEvent ae)
{
```

```
        long n=Integer.parseInt(tf1.getText());
        long f=1;
        while(n !=0)
        {
            f=f*n;
            n--;
        }
        tf2.setText(String.valueOf(f));
    }
}
```

Output:



Result :

Thus the program for finding the factorial of a number using Applet has been executed successfully and the output was verified.

Ex. No : 10 **Socket Programming**
Date :

Aim:

To write a program to implement socket programming in Java.

Algorithm:

Step1: Create a server socket and bind it to a specific port number
Step2: Listen for a connection from the client and accept it. This results in a client socket is created for the connection.
Step3: Read data from the client via an InputStream obtained from the client socket.
Step4: Send data to the client via the client socket's OutputStream.
Step5: Close the connection with the client.
Step6: Stop the program

Program:

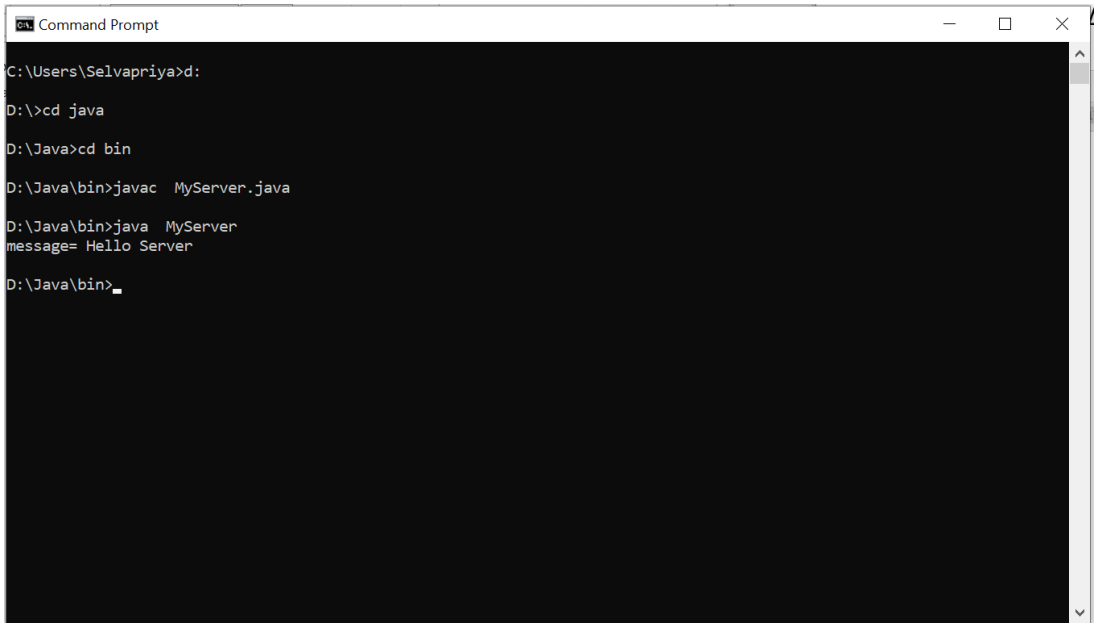
Server:

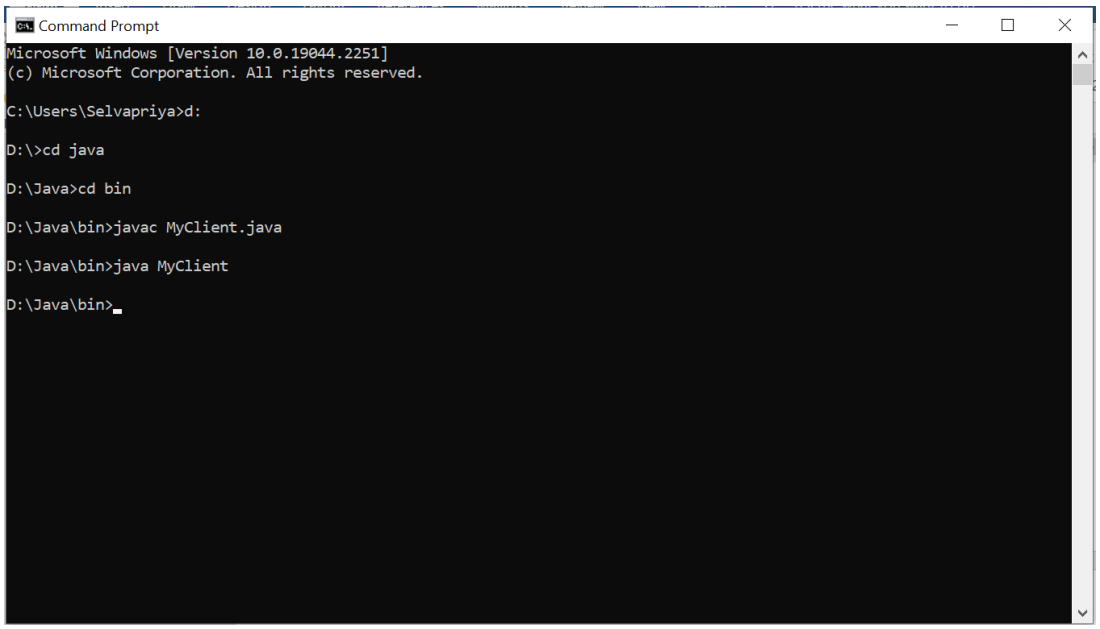
```
import Java.io.*;
import Java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Client:

```
import Java.io.*;
import Java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Output:





Result:

Thus the program to implement socket programming in Java has been executed successfully and the output was verified.