**TUTORIAL NO : 1**

**Q) Implement calculator program performing operations as addition, subtraction, multiplication ,division considering input in form of expression.**

**Theory :**

**a ) Monolithic Architecture :**

In software engineering , a **monolithic application** describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.

In software engineering, a monolithic application describes a software application which is designed without modularity. Modularity is desirable, in general, as it supports reuse of parts of the application logic and also facilitates maintenance by allowing repair or replacement of parts of the application without requiring wholesale replacement **.**

**b) Client Server Architecture :**

Client system handles both Presentation and Application layers and Server system handles Database layer. It is also known as client server application. The communication takes place between the Client and the Server. Client system sends the request to the Server system and the Server system processes the request and sends back the data to the Client System.

The client/server architectural style describes distributed systems that involve a separate client and server system, and a connecting network. The simplest form of client/server system involves a server application that is accessed directly by multiple clients, referred to as a 2-Tier architectural style.

The main benefits of the client/server architectural style are:

* **Higher security**. All data is stored on the server, which generally offers a greater control of security than client machines.
* **Centralized data access**. Because data is stored only on the server, access and updates to the data are far easier to administer than in other architectural styles.
* **Ease of maintenance**. Roles and responsibilities of a computing system are distributed among several servers that are known to each other through a network. This ensures that a client remains unaware and unaffected by a server repair, upgrade, or relocation.

**Class Diagram :**

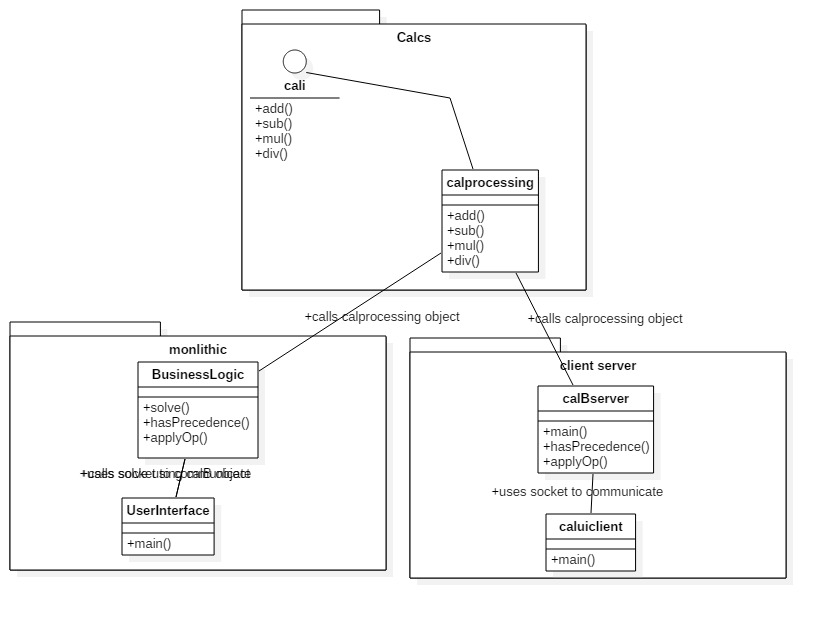


Fig : class diagram for monolithic and client server architecture

**Implementation :**

**Monolithic Architecture :**

//Interface

**publicinterface** cali {

**publicdouble** add(**double** a,**double** b);

**publicdouble** sub(**double** a,**double** b);

**publicdouble** mul(**double** a,**double** b);

**publicdouble** div(**double** a,**double** b);

}

//Bussiness class

**publicclass** calB **implements** cali {

**publicdouble** add(**double** a,**double** b)

{

**return** a+b;

}

**publicdouble** sub(**double** a,**double** b){**return** a-b;}

**publicdouble** mul(**double** a,**double** b){**return** a\*b;}

**publicdouble** div(**double** a,**double** b){**return** a/b;}

}

//User Interface class

**import** java.util.Scanner;

**import** java.util.Stack;

**public class** calui{

**public static void** main(String args[]){

System.*out*.println("Enter the expression:");

Scanner s=**new** Scanner(System.*in*);

String expression=s.nextLine();

**char**[] tokens = expression.toCharArray();

// Stack for numbers

Stack<Double> values = **new** Stack<Double>();

// Stack for Operators

Stack<Character> ops = **new** Stack<Character>();

**for** (**int** i = 0; i < tokens.length; i++)

{

// Current token is a number, push it to stack for numbers

**if** (tokens[i] >= '0'&& tokens[i] <= '9')

{

StringBuffer sbuf = **new** StringBuffer();

// There may be more than one digits in number

**while** (i < tokens.length&& tokens[i] >= '0'&& tokens[i] <= '9')

sbuf.append(tokens[i++]);

values.push(Double.*parseDouble*(sbuf.toString()));

}

**elseif** (tokens[i] == '(')

ops.push(tokens[i]);

**elseif** (tokens[i] == ')')

{

**while** (ops.peek() != '(')

values.push(*applyOp*(ops.pop(), values.pop(), values.pop()));

ops.pop();

}

**elseif** (tokens[i] == '+' || tokens[i] == '-' ||

tokens[i] == '\*' || tokens[i] == '/')

{

// While top of 'ops' has same or greater precedence to current

// token, which is an operator. Apply operator on top of 'ops'

// to top two elements in values stack

**while** (!ops.empty() &&*hasPrecedence*(tokens[i], ops.peek()))

values.push(*applyOp*(ops.pop(), values.pop(), values.pop()));

ops.push(tokens[i]);

}

}

// Entire expression has been parsed at this point, apply remaining ops to remaining values

**while** (!ops.empty())

values.push(*applyOp*(ops.pop(), values.pop(), values.pop()));

// Top of 'values' contains result, return it

System.*out*.println(values.pop());

s.close();

**return**;

}

// Returns true if 'op2' has higher or same precedence as 'op1',

// otherwise returns false.

**publicstaticboolean** hasPrecedence(**char** op1, **char** op2)

{

**if** (op2 == '(' || op2 == ')')

**returnfalse**;

**if** ((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-'))

**returnfalse**;

**else**

**returntrue**;

}

// A utility method to apply an operator 'op' on operands 'a'

// and 'b'. Return the result.

**publicstatic** Double applyOp(**char** op, Double b, Double a)

{

calB cal=**new** calB();

**switch** (op)

{

**case**'+':

**return** cal.add(a,b); // calling the bussiness class

**case**'-':

**return** cal.sub(a, b);

**case**'\*':

**return** cal.mul(a, b);

**case**'/':

**if** (b == 0)

**thrownew**

UnsupportedOperationException("Cannot divide by zero");

**return** cal.div(a, b);

}

**return** 0.00;

}

}

**Client Server Architecture :**

**Server code :**

|  |
| --- |
| package clientServer; |
|  |

|  |
| --- |
|  |
|  |
|  |
|  |
| //Java program to illustrate Server Side Programming |
|  |
|  |
| //for Simple Calculator using TCP |
|  |
|  |
|  |
|  |
|  |
| import java.io.DataInputStream; |
|  |
|  |
| import java.io.DataOutputStream; |
|  |
|  |
| import java.io.IOException; |
|  |
|  |
| import java.net.ServerSocket; |
|  |
|  |
| import java.net.Socket; |
|  |
|  |
| import java.util.Scanner; |
|  |
|  |
| import java.util.Stack; |
|  |
|  |
| import calcs.calprocessing; |
|  |
|  |
|  |
|  |
|  |
| public class calBserver { |
|  |
|  |
| private static int portNum = 6666; |
|  |
|  |
|  |
|  |
|  |
| public static void main(String args[]) throws IOException { |
|  |
|  |
|  |
|  |
|  |
| // Step 1: Establish the socket connection. |
|  |
|  |
| System.out.println("Server up and running at port " + portNum); |
|  |
|  |
| ServerSocket serverSock = new ServerSocket(portNum); |
|  |
|  |
| Socket sock = serverSock.accept(); |
|  |
|  |
| Scanner s = new Scanner(System.in); |
|  |
|  |
|  |
|  |
|  |
| DataInputStream dis = new DataInputStream(sock.getInputStream()); |
|  |
|  |
| DataOutputStream dos = new DataOutputStream(sock.getOutputStream()); |
|  |
|  |
| System.out.println("Ready to take input!"); |
|  |
|  |
|  |
|  |
|  |
| // Step 2: Processing the request. |
|  |
|  |
| while (true) { |
|  |
|  |
| // wait for input |
|  |
|  |
| String input = (String)dis.readUTF(); |
|  |
|  |
|  |
|  |
|  |
| if (input.equals("bye")) |
|  |
|  |
| break; |
|  |
|  |
|  |
|  |
|  |
| char[] tokens = input.toCharArray(); |
|  |
|  |
| // Stack for numbers |
|  |
|  |
| Stack<Double> values = new Stack<Double>(); |
|  |
|  |
|  |
|  |
|  |
| // Stack for Operators |
|  |
|  |
| Stack<Character> ops = new Stack<Character>(); |
|  |
|  |
|  |
|  |
|  |
| for (int i = 0; i < tokens.length; i++) { |
|  |
|  |
| // Current token is a number, push it to stack for numbers |
|  |
|  |
| if (tokens[i] >= '0' && tokens[i] <= '9') { |
|  |
|  |
| StringBuffer sbuf = new StringBuffer(); |
|  |
|  |
| // There may be more than one digits in number |
|  |
|  |
| while (i < tokens.length && tokens[i] >= '0' && tokens[i] <= '9') |
|  |
|  |
| sbuf.append(tokens[i++]); |
|  |
|  |
| values.push(Double.parseDouble(sbuf.toString())); |
|  |
|  |
| } else if (tokens[i] == '(') { |
|  |
|  |
| ops.push(tokens[i]); |
|  |
|  |
| } else if (tokens[i] == ')') { |
|  |
|  |
| while (ops.peek() != '(') |
|  |
|  |
| values.push(applyOp(ops.pop(), values.pop(), values.pop())); |
|  |
|  |
| ops.pop(); |
|  |
|  |
| } else if (tokens[i] == '+' || tokens[i] == '-' || tokens[i] == '\*' || tokens[i] == '/') { |
|  |
|  |
| // While top of 'ops' has same or greater precedence to current |
|  |
|  |
| // token, which is an operator. Apply operator on top of 'ops' |
|  |
|  |
| // to top two elements in values stack |
|  |
|  |
| while (!ops.empty() && hasPrecedence(tokens[i], ops.peek())) |
|  |
|  |
| values.push(applyOp(ops.pop(), values.pop(), values.pop())); |
|  |
|  |
| ops.push(tokens[i]); |
|  |
|  |
| } |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| // Entire expression has been parsed at this point, apply remaining ops to remaining values |
|  |
|  |
| while (!ops.empty()) |
|  |
|  |
| values.push(applyOp(ops.pop(), values.pop(), values.pop())); |
|  |
|  |
|  |
|  |
|  |
| // Top of 'values' contains result, return it |
|  |
|  |
| System.out.println(values.peek()); |
|  |
|  |
| dos.writeUTF(Double.toString(values.pop())); |
|  |
|  |
| dos.flush(); |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| dis.close(); |
|  |
|  |
| dos.close(); |
|  |
|  |
| sock.close(); |
|  |
|  |
| serverSock.close(); |
|  |
|  |
| s.close(); |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| public static boolean hasPrecedence(char op1, char op2) { |
|  |
|  |
| if (op2 == '(' || op2 == ')') |
|  |
|  |
| return false; |
|  |
|  |
| if ((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-')) |
|  |
|  |
| return false; |
|  |
|  |
| Else |
|  |
|  |
| return true; |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| public static Double applyOp(char op, Double b, Double a) { |
|  |
|  |
| calprocessing cal=new calprocessing(); |
|  |
|  |
| switch (op) { |
|  |
|  |
| case '+': |
|  |
|  |
| return cal.add(a,b); // calling the bussiness class |
|  |
|  |
| case '-': |
|  |
|  |
| return cal.sub(a, b); |
|  |
|  |
| case '\*': |
|  |
|  |
| return cal.mul(a, b); |
|  |
|  |
| case '/': |
|  |
|  |
| if (b == 0) |
|  |
|  |
| throw new UnsupportedOperationException("Cannot divide by zero"); |
|  |
|  |
| return cal.div(a, b); |
|  |
|  |
| } |
|  |
|  |
| return 0.00; |
|  |

|  |
| --- |
| } |

|  |
| --- |
| } |

**Client code :**

|  |
| --- |
| package clientServer; |
|  |

|  |
| --- |
|  |
| //Java program to illustrate Client Side Programming |
|  |
|  |
| //for Simple Calculator using TCP |
|  |
|  |
| import java.io.DataInputStream; |
|  |
|  |
| import java.io.DataOutputStream; |
|  |
|  |
| import java.io.IOException; |
|  |
|  |
| import java.net.Socket; |
|  |
|  |
| import java.util.Scanner; |
|  |
|  |
|  |
|  |
|  |
| public class caluiclient { |
|  |
|  |
| public static void main(String[] args) throws IOException |
|  |
|  |
| { |
|  |
|  |
| // InetAddress ip = InetAddress.getLocalHost(); |
|  |
|  |
| Scanner sc = new Scanner(System.in); |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| int portNum = 6666; |
|  |
|  |
| // Step 1: Open the socket connection. |
|  |
|  |
| Socket s = new Socket("127.0.0.1", portNum); |
|  |
|  |
|  |
|  |
|  |
| // Step 2: Communication-get the input and output stream |
|  |
|  |
| DataInputStream dis = new DataInputStream(s.getInputStream()); |
|  |
|  |
| DataOutputStream dos = new DataOutputStream(s.getOutputStream()); |
|  |
|  |
|  |
|  |
|  |
| while (true) |
|  |
|  |
| { |
|  |
|  |
| // Enter the equation in the form- |
|  |
|  |
| // "operand1 operation operand2" |
|  |
|  |
| System.out.print("Enter the expression :"); |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| String inp = sc.nextLine(); |
|  |
|  |
|  |
|  |
|  |
| if (inp.equals("bye")) { |
|  |
|  |
| // tell server to close connection |
|  |
|  |
| dos.writeUTF("bye"); |
|  |
|  |
| dos.flush(); |
|  |
|  |
| break; |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| // send the equation to server |
|  |
|  |
| dos.writeUTF(inp); |
|  |
|  |
| dos.flush(); |
|  |
|  |
|  |
|  |
|  |
| // wait till request is processed and sent back to client |
|  |
|  |
| String ans = (String)dis.readUTF(); |
|  |
|  |
| System.out.println("Answer=" + ans); |
|  |
|  |
| } |
|  |
|  |
|  |
|  |
|  |
| dis.close(); |
|  |
|  |
| dos.close(); |
|  |
|  |
| sc.close(); |
|  |
|  |
| s.close(); |
|  |
|  |
| } |
|  |

|  |
| --- |
|  |

|  |
| --- |
| } |