**TUTORIAL NO. 01**

**AIM:** Implement calculator program performing operation as addition, subtraction, multiplication, division considering input in form of expression.

**THEORY:**

* **Design assumptions:**

1. **Monolithic architecture:**
2. A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means composed all in one piece.
3. Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent rather than loosely coupled as is the case with modular software programs.
4. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled.
5. there are benefits to monolithic architectures as well. Monolithic programs typically have better throughput than modular approaches, such as the micro service architecture (MSA) and they can be easier to test and debug because, with fewer elements there are fewer variables that come into play.
6. **Advantages of monolithic architecture:**

**1. Faster Initial Development:**

With one application, it would be relatively easy to add additional features, especially when the application is relatively small.  Several features were added to the 80,000 Hours codebase relatively easily (it would have taken more code to have separate applications for each one).

1. **Little User Confusion:**

Users wouldn't have to learn about different applications, but would be focussed towards one application.

1. **Improved Integration:**

Features could integrate with each other well and easily, as there is only one user table.

1. **User Interface Similarity:**

All of the pieces of the application would look very similar, so it would be obvious it's all part of one system.

**Disadvantages:**

1. **Substantially Less Iteration:**

The larger a website is, the more difficult it is to change it.  It would be incredibly tough, for example, to change the theme or UI or a monolith application.  The means that we would have significantly less experimentation. This is one reason why the 80,000 Hours social network was put on hold after the redesign; it would have vastly increased the time to actually make that happen.

1. **Maintenance:**

The larger a website is, the more difficult it becomes to maintain the entire thing.  Maintenance costs may go up exponentially with site size.  This is one reason why many startups, with large amounts of funding (1-20 million dollars) have relatively simple websites.  Large ones, especially large ones with large feature sets, typically don't have good reputations for stability.

1. **Power Centralization:**

Obviously if there's one monolith application, it would ultimately be owned by one person or organization.  If all application uses would be put into this application, then this is a lot of power to trust in one organization.  There's currently a decent level of distrust and conflicting goals between EA organizations.  Thus, to have one EA organization in charge of all EA apps (one large one) would present a situation that could create controversy.

1. **Client-Server architecture:**
2. Client-server (C/S) or "two-tier" architecture (in comparison with a peer-to-peer architecture) has two separate types of nodes on the network: servers, that store information and clients, that send requests for information to servers.
3. Usually, but not always, a client computer and a server computer are two separate devices. A server computer contains large amounts of memory and disk space, while client computers features graphic user interface to support the display of data stored on server. There are many different types of *client/server software*however their basic architecture remains the same.
4. **Client/Server Benefits:**

Client/server approach to networking has proven to be a cost-effective way to share data between tens or hundreds of computers. Considering the client/server computing in terms of a manager-employee relationship, the following list outlines some of the benefits and drawbacks of client/server solutions.

1. **Centralized Information Storage:**

The server stores the data and coordinates the access to information and its modification. This helps to keep the data consistent and up-to-date, even when multiple users/clients are working with it simultaneously.

1. **Delegation:**

Managers or team leaders have more information, experience and knowledge about the company and day-to-day operations. Their deep understanding of the business processes, priorities, strategy, goals, and important tasks allows them to easily share information as needed and delegate work to their employees.

1. **Focus:**

The employees or team members may have less knowledge and experience or their vision of the strategy and goals is far from clear. By using [client/server software](http://en.wikipedia.org/wiki/Client%E2%80%93server_model) they have more focused tasks and also they might get a clear picture of company strategy if their team leaders provide them with access to such information.

1. **Collaboration:**

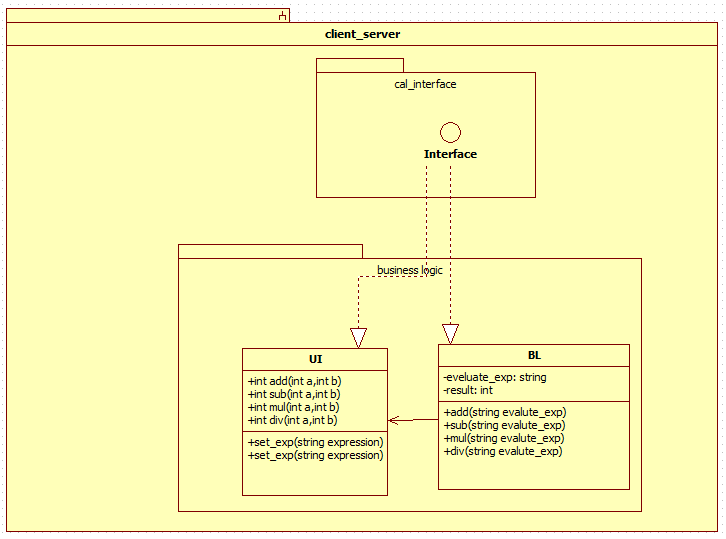
This is the essential aspect of how client/server computing works. It lets the manager plan, assign and notify the team members of their tasks. Once finished with their work, employees report the results back to the manager for verification.

1. **Security:**

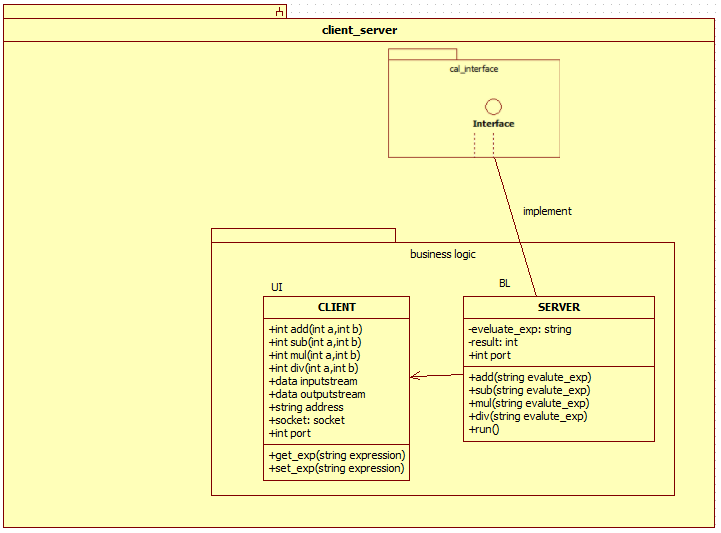
Very often the database is securely locked away from unauthorized access and its data is denied 'view' and/or 'edit' permission, that prevents violations from outside and inside the office.

* **UML Design Architecture:**

1. **Monolithic:**



1. **Client- Server:**



* **Code:**

1. **Calculator.java-**

|  |
| --- |
| import java.io.\*; |
|  | import java.util.Scanner; |
|  | import java.util.StringTokenizer; |
|  | class UI{ |
|  | int a; |
|  | int b; |
|  |  |
|  | void setij(int x, int y){ |
|  | a = x; |
|  | b = y; |
|  | } |
|  | } |
|  |  |
|  | class BI extends UI{ |
|  | int r ; |
|  | void add(){ |
|  | r = a + b; |
|  | } |
|  | void sub(){ |
|  | r = a - b; |
|  | } |
|  | void mul(){ |
|  | r = a \* b; |
|  | } |
|  | void div(){ |
|  | r = a / b; |
|  | } |
|  | } |
|  |  |
|  | class Calculator{ |
|  | public static void main(String args[]){ |
|  | BI obj = new BI(); |
|  | Scanner sc = new Scanner(System.in); |
|  | int result; |
|  | int op1; |
|  | int op2; |
|  |  |
|  | while(true){ |
|  |  |
|  | System.out.print("Enter the equation in the form: "); |
|  | System.out.println("'operand operator operand'"); |
|  |  |
|  | String inp = sc.nextLine(); |
|  | if (inp.equals("bye")) |
|  | break; |
|  | StringTokenizer st = new StringTokenizer(inp); |
|  |  |
|  | op1 = Integer.parseInt(st.nextToken()); |
|  | String operation = st.nextToken(); |
|  | op2 = Integer.parseInt(st.nextToken()); |
|  | obj.setij(op1, op2); |
|  | if (operation.equals("+")) |
|  | { |
|  | obj.add(); |
|  | System.out.println("Result : " +obj.r); |
|  | } |
|  |  |
|  | else if (operation.equals("-")) |
|  | { |
|  | obj.sub(); |
|  | System.out.println("Result : " +obj.r); |
|  | } |
|  | else if (operation.equals("\*")) |
|  | { |
|  | obj.mul(); |
|  | System.out.println("Result : " +obj.r); |
|  | } |
|  | else |
|  | { |
|  | obj.div(); |
|  | System.out.println("Result : " +obj.r); |
|  | } |
|  | } |
|  |  |
|  | } |
|  | } |

1. **Server.java-**

import java.net.\*;

import java.io.\*;

import java.text.\*;

import java.util.\*;

class calculator

{

Double calculate(String exp){

Stack<Integer> optmp = new Stack<Integer>();

Stack<Double> valtmp = new Stack<Double>();

Stack<Integer> op = new Stack<Integer>();

Stack<Double> val = new Stack<Double>();

String input = exp;

input = "0" + input;

input = input.replaceAll("-","+-");

String num = "";

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

{

char ch = input.charAt(i);

if (ch == '-')

num = "-" + num;

else if (ch != '+' && ch != '\*' && ch != '/')

num = num + ch;

else

{

val.push(Double.parseDouble(num));

op.push((int)ch);

num = "";

}

}

val.push(Double.parseDouble(num));

char operators[] = {'/','\*','+'};

for (int i = 0; i < 3; i++)

{

boolean it = false;

while (!op.isEmpty())

{

int optr = op.pop();

double v1 = val.pop();

double v2 = val.pop();

if (optr == operators[i])

{

if (i == 0)

{

valtmp.push(v2 / v1);

it = true;

break;

}

else if (i == 1)

{

valtmp.push(v2 \* v1);

it = true;

break;

}

else if (i == 2)

{

valtmp.push(v2 + v1);

it = true;

break;

}

}

else

{

valtmp.push(v1);

val.push(v2);

optmp.push(optr);

}

}

while (!valtmp.isEmpty())

val.push(valtmp.pop());

while (!optmp.isEmpty())

op.push(optmp.pop());

if (it)

i--;

};

return val.pop();

}

}

public class server1

{

private Socket socket = null;

private ServerSocket server = null;

private DataInputStream in = null;

private DataOutputStream output = null;

public server1(int port)

{

try

{

server = new ServerSocket(port);

while(true){

socket=null;

System.out.println("Server started");

System.out.println("Waiting for a client ...");

socket = server.accept();

System.out.println("Client accepted");

// takes input from the client socket

in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

output= new DataOutputStream(socket.getOutputStream());

System.out.println("Assigning new thread for this client");

// create a new thread object

Thread t = new ClientHandler(socket, in, output);

// Invoking the start() method

t.start();

}

}catch (Exception e){

e.printStackTrace();

}

}

public static void main(String args[])

{

server1 server = new server1(5000);

}

}

class ClientHandler extends Thread {

final DataInputStream dis;

final DataOutputStream dos;

final Socket s;

// Constructor

public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)

{

this.s = s;

this.dis = dis;

this.dos = dos;

}

@Override

public void run()

{

String line = "";

calc cal = new calc();

// reads message from client until "Over" is sent

try

{

text = dis.readUTF();

double answer=0.0;

while(!text.equals("done")){

answer= cal.calculate(text);

dos.writeUTF(""+answer);

text = dis.readUTF();

}

}

catch(IOException i)

{

System.out.println(i);

}

try

{

System.out.println("Closing connection");

// close connection

s.close();

dis.close();

// closing resources

this.dis.close();

this.dos.close();

}catch(IOException e){

e.printStackTrace();

}

}

}

1. **Client.java:**

|  |
| --- |
|  |

import java.util.Scanner;

import java.net.Socket;

import java.io.BufferedInputStream;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.UnknownHostException;

import java.util.\*;

public class client

{

private Socket socket = null;

private DataInputStream input = null;

private DataInputStream in = null;

private DataOutputStream out = null;

public client(String address, int port)

{

try

{

socket = new Socket(address, port);

System.out.println("Client Connected");

input = new DataInputStream(System.in);

out = new DataOutputStream(socket.getOutputStream());

in=new DataInputStream(new BufferedInputStream(socket.getInputStream()));

}

catch(UnknownHostException u)

{

System.out.println(u);

}

catch(IOException i)

{

System.out.println(i);

}

String text = "";

Scanner sc=new Scanner(System.in);

Stack<Double> res = new Stack<Double>();

try

{

System.out.println("Enter an expression:");

while(!text.equals("done"))

{

text = sc.nexttext();

out.writeUTF(text);

double answer=0.0;

answer = Double.parseDouble(in.readUTF());

System.out.println("Answer is "+answer);

res.push(answer);

}

}

catch(IOException i)

{

System.out.println(i);

}

try

{

System.out.println("Your all answers are");

while (!res.isEmpty()){

System.out.println(res.pop());

}

input.close();

out.close();

socket.close();

}

catch(IOException i)

{

System.out.println(i);

}

}

public static void main(String args[])

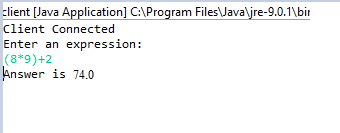
{

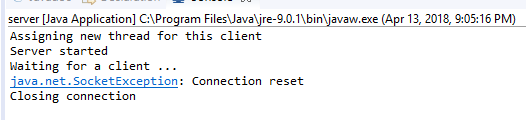
client cli = new client("127.0.0.1", 5000);

}

}

**OUTPUT:**

****



**OBERVATION:**

Using client server and monolithic architecture we can build calculator. But when we implement code for client server and monolithic architecture, we observe that both architecture follow the simple calculator process so in both architectures we can import the package of normal calculator. The monolithic architecture doesn’t allow to communicate via server so we can solve this issue using client-server architecture.