Group members:

161081971 – Apeksha Barve

161081977 – Sayali Patil

161081978 – Pragati Patil

161081982 – Bhagyahri Kamble

Tutorial No. 02

Aim:

Theory:

1. “multi-threading” is considered to be synonymous with similar sounding terms like “multi-programming” or “multi-tasking”.
2. Actually, multi-threading defines the capability of a software, an operating system, a process or a hardware component to manage more than one user at a time. Such an application can accept multiple requests by one user or many users without running multiple instances of the program on the computer.
3. A socket is an end-point of the connection between two programs running on a network. This end-point is required to establish a communication link between a server program and a client program. These programs run on different computers which are called local and remote computers. But, these two programs can also be run on a single computer.
4. advantages of a multi-threaded server?  
   The main advantages of using a multi-threaded server over a single-threaded one in Java are given below:
5. It is very quick and doesn’t obstruct the user, as every thread is independent of each other and the program. Thus, the server can respond to many queries at once.
6. As threads are independent of each other, they do not block the work of other threads if exceptions occur in the thread.
7. As it can perform many operations at once, it can save lots of time. If a process runs for a long time then single-threaded servers may become unresponsive, but this is not the case in multi-threaded servers.

# Requirements Specification:

Calculator using multi-threaded server is basically a medium of connecting clients and server which is useful to calculate some basic operations.

## User Interfaces

The user can calculate the result by entering the expression on terminal. When client send expression as input to server then server calculate the result and return or display the final answer.

**Response Sequence**

1. Server must be on.
2. When client issue open command via server then client can connect with server.
3. Server send confirmation to particular client
4. Then client send expression as input to server.
5. Server calculate the result and display it on client terminal.
6. Then connection get close.

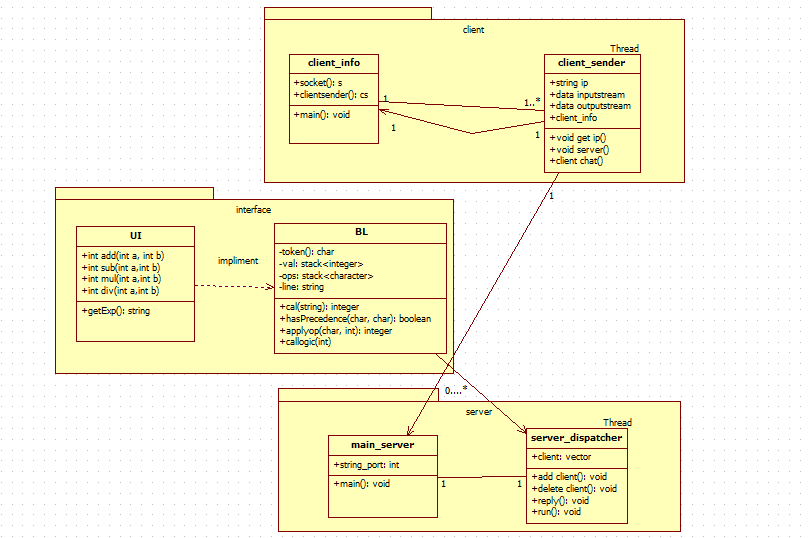
## Hardware Interfaces

The multithreaded client server calculator need some basic hardware requirements like our code is in JAVA language then this language must be supported to windows.

## Software Interfaces

The multithreaded client server program is written into JAVA language. So the software needed to execute this program is eclipse, net beans. The program must have supported to JAVA libraries.

Class Diagram:



**client \_side:**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.InetAddress;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.Scanner;

public class Calc\_Client

{

public static void main(String[] args) throws IOException

{

try {

InetAddress ip = InetAddress.getLocalHost();

int port = 4444;

SendThread sendThread = new SendThread(sock);

Thread thread = new Thread(sendThread);thread.start();

RecieveThread recieveThread = new RecieveThread(sock);

Thread thread2 =new Thread(recieveThread);thread2.start();

} catch (Exception e) {System.out.println(e.getMessage());}

}

Scanner sc = new Scanner(System.in);

// Step 1: Open the socket connection.

Socket s = new Socket(ip, port);

// 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 equation in the form: ");

System.out.println("'operand operator operand'");

String inp = sc.nextLine();

if (inp.equals("bye"))

break;

// send the equation to server

dos.writeUTF(inp);

// wait till request is processed and sent back to client

String ans = dis.readUTF();

System.out.println("Answer=" + ans);

}

}

}

**server\_side :**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.StringTokenizer;

public class Calc\_Server

{

public static void main(String args[]) throws IOException

{

// Step 1: Establish the socket connection.

ServerSocket ss = new ServerSocket(4444);

Socket s = ss.accept();

RecieveFromClientThread recieve = new RecieveFromClientThread(clientSocket);

Thread thread = new Thread(recieve);

thread.start();

SendToClientThread send = new SendToClientThread(clientSocket);

Thread thread2 = new Thread(send);

thread2.start();

}

}

class RecieveFromClientThread implements Runnable

{

Socket clientSocket=null;

BufferedReader brBufferedReader = null;

public RecieveFromClientThread(Socket clientSocket)

{

this.clientSocket = clientSocket;

}

// Step 2: Processing the request.

DataInputStream dis = new DataInputStream(s.getInputStream());

DataOutputStream dos = new DataOutputStream(s.getOutputStream());

while (true)

{

// wait for input

String input = dis.readUTF();

if(input.equals("bye"))

break;

System.out.println("Equation received:-" + input);

int result;

// Use StringTokenizer to break the equation into operand and

// operation

StringTokenizer st = new StringTokenizer(input);

int oprnd1 = Integer.parseInt(st.nextToken());

String operation = st.nextToken();

int oprnd2 = Integer.parseInt(st.nextToken());

// perform the required operation.

if (operation.equals("+"))

{

result = oprnd1 + oprnd2;

}

else if (operation.equals("-"))

{

result = oprnd1 - oprnd2;

}

else if (operation.equals("\*"))

{

result = oprnd1 \* oprnd2;

}

else

{

result = oprnd1 / oprnd2;

}

System.out.println("Sending the result...");

// send the result back to the client.

dos.writeUTF(Integer.toString(result));

}

}

}