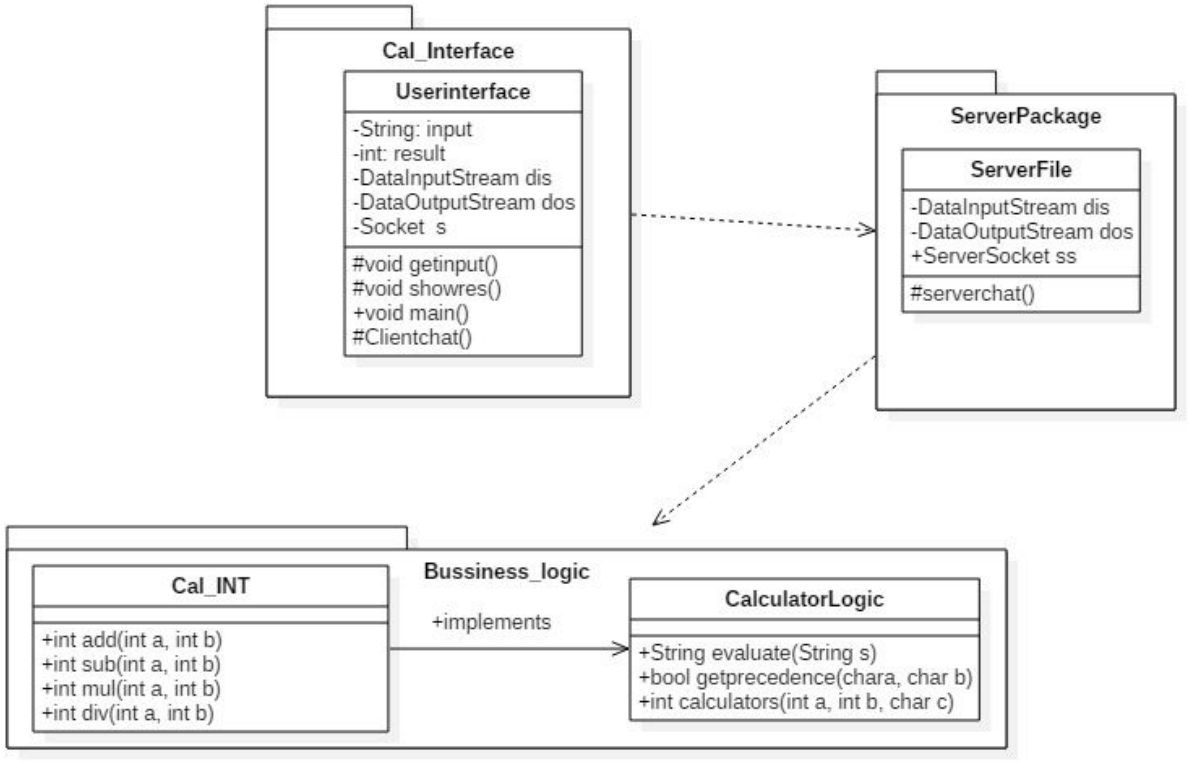
Tutorial No.2

Q. Apply server side Multithreading in calculator system

Class diagram:



Implementation:

//Client side

```
package ClientPackage;
import java.io.*;
import java.net.*;
import java.util.*;

public class ClientFile {

        Socket s;
   DataInputStream din;
   DataOutputStream dout;
```

```
    public static void main(String as[])
    {
       new ClientFile();
    }


    public ClientFile()
    {
        try
        {

            s=new Socket("localhost",10);
            //System.out.println(s);
            din= new DataInputStream(s.getInputStream());
            dout= new DataOutputStream(s.getOutputStream());
            ClientChat();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
     }
     public void ClientChat() throws IOException
     {
            String choice;
            do{

                    Scanner scan=new Scanner(System.in);
                    System.out.println("Enter the Expression : ");
                    String exp=scan.nextLine();
                    dout.writeUTF(exp);
                    System.out.println(din.readUTF());
                    String servermsg=din.readUTF();
                    System.out.println(servermsg);
                    choice=scan.next();
                    dout.writeUTF(choice);
                    dout.flush();
            }while(choice.equals("Y"));
    }
}
```

Server side:

```
package ServerPackage;
import java.io.*;
import java.net.*;
import java.util.Stack;

public class ServerFile {
	public static void main(String args[]){
		try{
			ServerSocket ss=new ServerSocket(10);
			while (true)
		{
		   Socket s = null;
		   try
		   {
		      s = ss.accept();
		      System.out.println("A new client is connected : " + s);
		      DataInputStream dis = new DataInputStream(s.getInputStream());
		      DataOutputStream dos = new DataOutputStream(s.getOutputStream());
		      System.out.println("Assigning new thread for this client");
		      Thread t = new Calculator(s, dis, dos);
		      t.start();
		   }
		   catch (Exception e){
		      s.close();
		      e.printStackTrace();
		   }
		}
		   }catch(Exception e){}
	}
}
class Calculator extends Thread{
	Socket s;
	DataInputStream din;
	DataOutputStream dos;
	public Calculator(Socket s,DataInputStream din,DataOutputStream dos){
		this.s=s;
		this.din=din;
		this.dos=dos;
	}
	public void run(){
		try{
```

```
                do{
                        String exp=din.readUTF();
                        dos.writeUTF(evaluate(exp)+"");
                        dos.writeUTF("Do you want to continue?");
                }while(din.readUTF().equals("Y"));
        }
        catch(Exception e){}
}
public static int evaluate(String expression)
{
        char[] tokens = expression.toCharArray();
        Stack<Integer> values = new Stack<Integer>();
        Stack<Character> ops = new Stack<Character>();
        for (int i = 0; i < tokens.length; i++)
        {
                if (tokens[i] == ' ')
                        continue;
                if (tokens[i] >= '0' && tokens[i] <= '9')
                {
                        StringBuffer sbuf = new StringBuffer();
                        while (i < tokens.length && tokens[i] >= '0' && tokens[i] <= '9')
                                sbuf.append(tokens[i++]);
                        values.push(Integer.parseInt(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 (!ops.empty() && hasPrecedence(tokens[i], ops.peek()))
                        values.push(applyOp(ops.pop(), values.pop(), values.pop()));
                        ops.push(tokens[i]);
                }
        }
        while (!ops.empty())
                values.push(applyOp(ops.pop(), values.pop(), values.pop()));
        return values.pop();
```

```
	}
	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 int applyOp(char op, int b, int a)
	{
		switch (op)
		{
		case '+':
			return a+b;
		case '-':
			return a-b;
		case '*':
			return a*b;
		case '/':
			if (b != 0)
				return b/a;
		}
		return 0;
	}
}
```