**Tutorial No. 2**

**Problem statement:**

Implement calculator program(client/server) using multithreaded server. Each user will be served by different threads. At the end of calculation, server will respond with all calculations done by specific user.

**Design Assumptions:**

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.  
  
Threads can be created by using two mechanisms :

* Extending the Thread class.
* Implementing the Runnable Interface

Example :

// Java code for thread creation by extending

// the Thread class

class MultithreadingDemo extends Thread

{

    public void run()

    {

        try

        {

            // Displaying the thread that is running

            System.out.println ("Thread " +

                  Thread.currentThread().getId() +

                  " is running");

        }

        catch (Exception e)

        {

            // Throwing an exception

            System.out.println ("Exception is caught");

        }

    }

}

// Main Class

public class Multithread

{

    public static void main(String[] args)

    {

        int n = 8; // Number of threads

        for (int i=0; i<8; i++)

        {

            MultithreadingDemo object = new MultithreadingDemo();

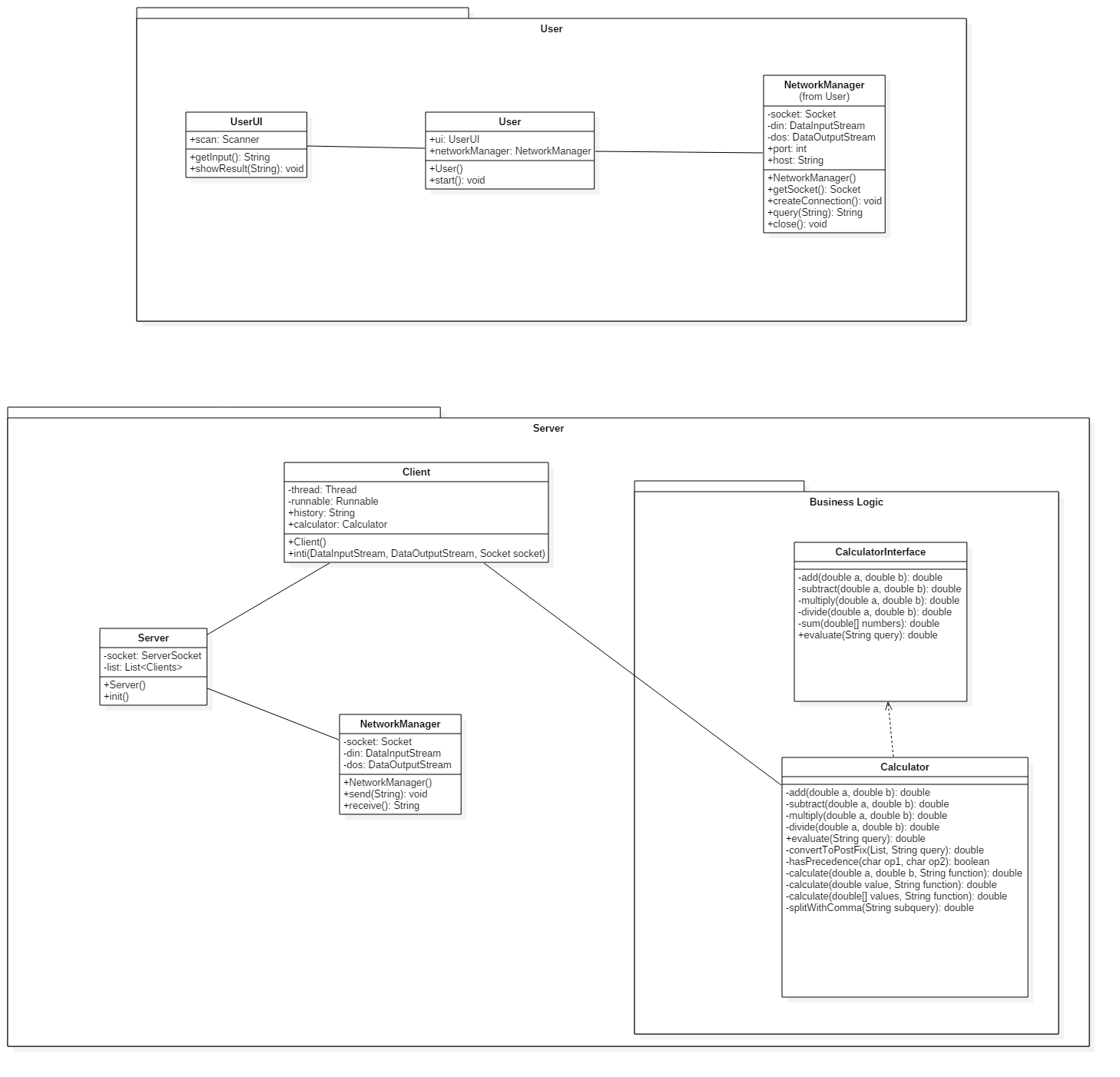
            object.start();

        }

    }

}

**Design Diagrams:**



**Code:**

**Calculator.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.server;

import java.util.ArrayList;

import java.util.List;

import java.util.Stack;

public class Calculator implements CalculatorInterface {

@Override

public double add(double a, double b) {

return a+b;

}

@Override

public double subtract(double a, double b) {

return a-b;

}

@Override

public double multiply(double a, double b) {

return a\*b;

}

@Override

public double divide(double a, double b) {

return a/b;

}

@Override

public double evaluate(String query) {

query = query.replaceAll("pi", "3.1415926535");

List<String> list = new ArrayList<>();

//query = normalize(query);

convertToPostFix(list, query);

return evaluate(list);

}

private double evaluate(List<String> list){

Stack<String> stack = new Stack<>();

for(int i=0; i<list.size(); i++){

if(list.get(i).equals("+")||list.get(i).equals("-")||list.get(i).equals("\*")||list.get(i).equals("/")){

double a = Double.parseDouble(stack.pop());

double b = Double.parseDouble(stack.pop());

stack.push(""+calculate(a,b,list.get(i).charAt(0)+""));

}

else{

stack.push(list.get(i));

}

}

return Double.valueOf(stack.pop());

}

private void convertToPostFix(List<String> list, String query) {

Stack<Character> operators = new Stack<>();

char[] tokens = query.toCharArray();

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

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

StringBuilder stringBuilder = new StringBuilder();

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

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

}

i--;

list.add(stringBuilder.toString());

}

else if(tokens[i] == '(')

operators.push('(');

else if (tokens[i] == ')'){

while (operators.peek() != '(')

list.add(operators.pop()+"");

operators.pop();

}

else if (tokens[i] == '+' || tokens[i] == '-' || tokens[i] == '\*' || tokens[i] == '/'){

while (!operators.empty() && hasPrecedence(tokens[i], operators.peek())){

list.add(operators.pop()+"");

}

operators.push(tokens[i]);

}

}

while (!operators.empty())

list.add(operators.pop()+"");

}

private boolean hasPrecedence(char op1, char op2) {

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

return false;

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

return false;

else

return true;

}

//calling arithmetic functions

private double calculate(double a, double b, String function){

switch (function){

case "+":

return add(a,b);

case "-":

return subtract(b,a);

case "\*":

return multiply(a,b);

case "/":

return divide(b,a);

}

return 0;

}

}

**CalculatorInterface.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.server;

public interface CalculatorInterface {

//basic

public double add(double a, double b);

public double subtract(double a, double b);

public double multiply(double a, double b);

public double divide(double a, double b);

public double evaluate(String query);

}

**Client.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.server;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class Client {

private Thread thread;

private Runnable runnable;

private String history;

private Calculator calculator;

public Client(DataInputStream dis, DataOutputStream dos, Socket socket) {

init(dis, dos, socket);

}

public void init(DataInputStream dis, DataOutputStream dos, Socket socket){

NetworkManager networkManager = new NetworkManager(dis, dos, socket);

calculator = new Calculator();

history = "";

runnable = () -> {

while (true) {

String query = null;

try {

query = networkManager.receive();

history += query + "\n";

System.out.println("Request from : " + networkManager.getPort());

} catch (IOException e) {

e.printStackTrace();

break;

}

System.out.println(query);

if(!query.equals("exit")){

String answer = calculator.evaluate(query) + "";

history += answer + "\n";

try {

networkManager.send(answer);

} catch (IOException e) {

e.printStackTrace();

}

}

else{

try {

networkManager.send(history);

break;

} catch (IOException e) {

e.printStackTrace();

break;

}

}

}

};

thread = new Thread(runnable);

thread.start();

}

}

**NetworkManager.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.server;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class NetworkManager {

final private DataInputStream dis;

final private DataOutputStream dos;

final private Socket socket;

public NetworkManager(DataInputStream dis, DataOutputStream dos, Socket socket){

this.dis = dis;

this.dos = dos;

this.socket = socket;

}

public int getPort(){

return socket.getPort();

}

public String receive() throws IOException{

String received;

received = dis.readUTF();

if (received.equals("Hello")) {

System.out.println("Working");

return "Hello";

}

else{

System.out.println(received);

return received;

}

}

public void send(String answer) throws IOException {

dos.writeUTF(answer);

}

}

**Server.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.server;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.List;

public class Server {

ServerSocket socket;

List<Client> clients = new ArrayList<>();

Calculator calculator;

public Server(){

calculator = new Calculator();

init();

}

public void init(){

try {

socket = new ServerSocket(8192);

} catch (IOException e) {

e.printStackTrace();

System.out.println("Failed to create Server!!");

}

while(true){

Socket s;

try{

s = socket.accept();

System.out.println("A new client has connected");

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

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

clients.add(new Client(dis, dos, s));

} catch (IOException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args){

new Server();

}

}

**NetworkManager.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.user;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class NetworkManager {

Socket socket = null;

DataInputStream din = null;

DataOutputStream dout = null;

int port;

String host;

public NetworkManager(String host, int port){

this.port = port;

this.host = host;

}

private Socket getSocket() throws IOException {

return new Socket(host, port);

}

public void createConnection(){

try {

socket = getSocket();

} catch (IOException e) {

e.printStackTrace();

System.out.println ("Failed to Create Socket");

return;

}

try {

din = new DataInputStream(socket.getInputStream());

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

} catch (IOException e) {

e.printStackTrace();

System.out.println("Failed to connect to the server");

return;

}

}

public String query(String query) throws IOException {

dout.writeUTF(query);

return din.readUTF();

}

public void close() {

try {

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**User.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.user;

import java.io.IOException;

import java.util.Scanner;

public class User {

Scanner scan = new Scanner(System.in);

UserUI ui;

public static void main(String args[]) {

new User();

}

public User() {

ui = new UserUI();

start();

}

public void start(){

NetworkManager networkManager = new NetworkManager("localhost", 8192);

networkManager.createConnection();

String query = "";

while(!query.equals("exit")){

query = ui.getInput();

if(!query.equals("")){

String ans;

try {

ans = networkManager.query(query);

} catch (IOException e) {

ui.showResult("Connection Reset");

networkManager.close();

return;

}

ui.showResult(ans);

}

}

}

}

**UserUI.java**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package com.mangnaik.yogesh.tutorial2.user;

import java.util.Scanner;

public class UserUI {

Scanner scan = new Scanner(System.in);

public String getInput(){

String query = scan.nextLine();

return query;

}

public void showResult(String answer){

System.out.println(answer);

}

}

**Observations:**

When creating multithreaded server we create a new thread for each client that connects to the server. Now if many clients connect to the server, the server may get overwhelmed. And if some users leave the server then we have to clear the resources it was using. Otherwise the resources of server will be wasted. One of the ways of reusing resources is using thread pools. It will make reusing of threads very easy.

When client is connected, a new object of serverclient is created and the thread for that client is started. The socket is started on a random port and thus we can have many ports. When client send a request the request is accepted by the server and it calculates the answer to the query and then it returns the answer. When client decides to disconnect from the server the server sends all the calculations done till now to the client.