**Tutorial 1**

**Problem statement:**

Implement calculator program

Operations :

1. Addition
2. Subtraction
3. Multiplication
4. Division

It takes input as string

1. Monolithic Architecture
2. Client – Server architecture

**Design Assumptions:**

**Monolithic Architecture**

In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), a monolithic application describes a single-tiered [software application](https://en.wikipedia.org/wiki/Application_software) in which the [user interface](https://en.wikipedia.org/wiki/User_interface) and data access code are combined into a single program from a single [platform](https://en.wikipedia.org/wiki/Platform_(computing)).

A monolithic application is self-contained, and independent from other computing applications. The design philosophy is that the application is responsible not just for a particular task, but can perform every step needed to complete a particular function. Today, some personal finance applications are monolithic in the sense that they help the user carry out a complete task, end to end, and are "private data silos" rather than parts of a larger system of applications that work together. Some [word processors](https://en.wikipedia.org/wiki/Word_processor) are monolithic applications. These applications are sometimes associated with [mainframe computers](https://en.wikipedia.org/wiki/Mainframe_computers).

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.

Modularity is achieved to various extents by different modularization approaches. Code-based modularity allows developers to reuse and repair parts of the application, but development tools are required to perform these maintenance functions (e.g. the application may need to be recompiled). Object-based modularity provides the application as a collection of separate executable files which may be independently maintained and replaced without redeploying the entire application (e.g. [Microsoft "dll" files](https://en.wikipedia.org/wiki/Dynamic-link_library); Sun/UNIX "shared object" files). Some object messaging capabilities allow object-based applications to be distributed across multiple computers (e.g. Microsoft COM+). Service-oriented architectures use specific communication standards/protocols to communicate between modules.

In its original use, the term "monolithic" described enormous main frame applications with no usable modularity. This – in combination with rapid increase in computational power and therefore rapid increase in the complexity of the problems which could be tackled by software – resulted in unmaintainable systems and the "[software crisis](https://en.wikipedia.org/wiki/Software_crisis)".

**Client Server architecture**

The client–server model is a [distributed application](https://en.wikipedia.org/wiki/Distributed_application) structure that partitions tasks or workloads between the providers of a resource or service, called [servers](https://en.wikipedia.org/wiki/Server_(computing)), and service requesters, called [clients](https://en.wikipedia.org/wiki/Client_(computing)). Often clients and servers communicate over a [computer network](https://en.wikipedia.org/wiki/Computer_network) on separate hardware, but both client and server may reside in the same system. A server [host](https://en.wikipedia.org/wiki/Host_(network)) runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are [Email](https://en.wikipedia.org/wiki/Email), [network printing](https://en.wikipedia.org/wiki/Network_printing), and the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web).

The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide. For example, a [web server](https://en.wikipedia.org/wiki/Web_server) serves [web pages](https://en.wikipedia.org/wiki/Web_page) and a [file server](https://en.wikipedia.org/wiki/File_server) serves [computer files](https://en.wikipedia.org/wiki/Computer_file). A shared resource may be any of the server computer's software and electronic components, from [programs](https://en.wikipedia.org/wiki/Computer_program) and [data](https://en.wikipedia.org/wiki/Data_(computing)) to [processors](https://en.wikipedia.org/wiki/Microprocessor) and [storage devices](https://en.wikipedia.org/wiki/Data_storage_device). The sharing of resources of a server constitutes a service.

Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer. Communication between servers, such as to synchronize data, is sometimes called [inter-server](https://en.wikipedia.org/wiki/Inter-server) or server-to-server communication.

**Example:**

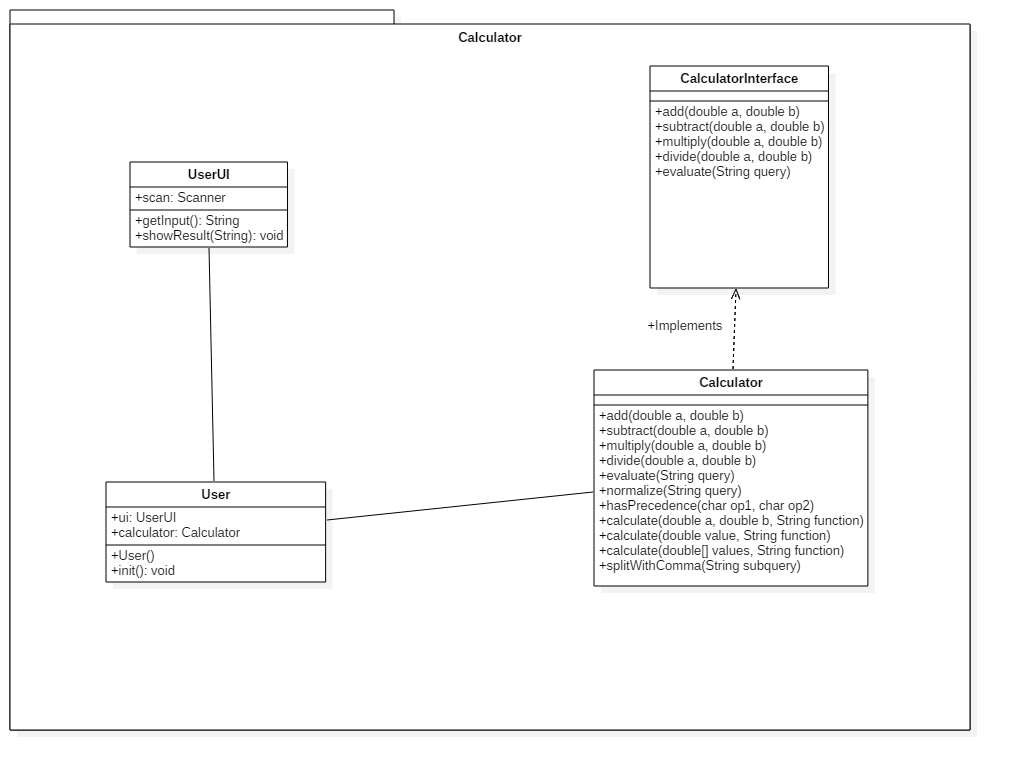
When a [bank](https://en.wikipedia.org/wiki/Bank) customer accesses [online banking](https://en.wikipedia.org/wiki/Online_banking) services with a [web browser](https://en.wikipedia.org/wiki/Web_browser) (the client), the client initiates a request to the bank's web server. The customer's [login](https://en.wikipedia.org/wiki/Login) credentials may be stored in a [database](https://en.wikipedia.org/wiki/Database), and the web server accesses the [database server](https://en.wikipedia.org/wiki/Database_server) as a client. An [application server](https://en.wikipedia.org/wiki/Application_server) interprets the returned data by applying the bank's [business logic](https://en.wikipedia.org/wiki/Business_logic), and provides the [output](https://en.wikipedia.org/wiki/Input/output) to the web server. Finally, the web server returns the result to the client web browser for display.

In each step of this sequence of client–server message exchanges, a computer processes a request and returns data. This is the request-response messaging pattern. When all the requests are met, the sequence is complete and the web browser presents the data to the customer.

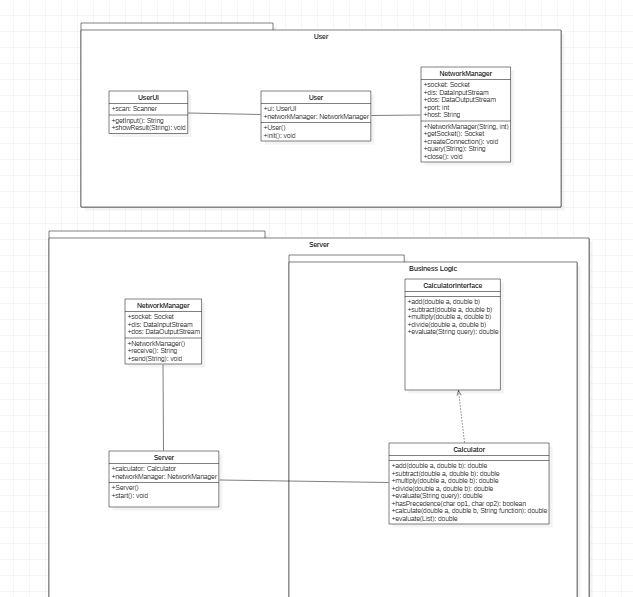
This example illustrates a [design pattern](https://en.wikipedia.org/wiki/Design_pattern) applicable to the client–server model: [separation of concerns](https://en.wikipedia.org/wiki/Separation_of_concerns).

**Design Diagrams:**

**Monolithic architecture:**

****

**Client server architecture:**

****

**Code:**

**Calculator monolithic :**

**Calculator.java**

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

package com.mangnaik.yogesh.calculator;

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;

}

}

**Calculator.java (interface)**

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

|  |  |
| --- | --- |
| package com.mangnaik.yogesh.calculator;  /\*\*  \* Created by Yogesh on 2/8/2018.  \*/  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);  } | |
|  | | |
|  | | |  |
|  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  | |  | |
|  |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |

**User.java**

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

package com.mangnaik.yogesh.calculator;

import java.io.IOException;

public class User {

UserUI ui;

Calculator calculator;

public static void main(String args[]) {

new User();

}

public User() {

ui = new UserUI();

calculator = new Calculator();

init();

}

private void init(){

String query = "";

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

query = ui.getInput();

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

double answer = calculator.evaluate(query);

ui.showResult(answer+"");

}

}

}

}

**Userui.java**

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

package com.mangnaik.yogesh.calculator;

import java.util.Scanner;

/\*\*

\* Created by Yogesh on 2/15/2018.

\*/

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);

}

}

**Client Server Architecture**

**Network Manger**

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

package com.mangnaik.yogesh.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");

}

}

public String query(String query) throws IOException {

dout.writeUTF(query);

String ans = din.readUTF();

return ans;

}

public void close() {

try {

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**User class**

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

package com.mangnaik.yogesh.user;

import java.io.IOException;

/\*\*

\* Created by Yogesh on 2/2/2018.

\*/

public class User {

UserUI ui;

NetworkManager networkManager;

public static void main(String args[]) {

new User();

}

public User() {

ui = new UserUI();

init();

}

private void init(){

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

networkManager.createConnection();

String query = "";

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

query = ui.getInput();

System.out.println("Query : " + query);

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

String ans;

try {

ans = networkManager.query(query);

} catch (IOException e) {

System.out.println("Connection Reset");

networkManager.close();

return;

}

ui.showResult(ans);

}

}

}

}

**UserUI class**

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

package com.mangnaik.yogesh.user;

import java.util.Scanner;

/\*\*

\* Created by Yogesh on 2/15/2018.

\*/

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);

}

}

**Calculator**

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

package com.mangnaik.yogesh.server;

import java.util.ArrayList;

import java.util.List;

import java.util.Stack;

/\*\*

\* Created by Yogesh on 2/8/2018.

\*/

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**

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

package com.mangnaik.yogesh.server;

/\*\*

\* Created by Yogesh on 2/8/2018.

\*/

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);

}

**Network Manager**

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

package com.mangnaik.yogesh.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 String receive(){

String received;

try {

received = dis.readUTF();

System.out.println("Received String : " + received);

return received;

} catch (IOException e) {

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

try {

socket.close();

return "";

} catch (IOException e1) {

e1.printStackTrace();

}

e.printStackTrace();

}

return "";

}

public void send(String answer){

try{

System.out.println("Sending String" + answer);

dos.writeUTF(answer);

}

catch(IOException e){

e.printStackTrace();

}

}

}

**Server.java**

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

package com.mangnaik.yogesh.server;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

/\*\*

\* Created by Yogesh on 2/8/2018.

\*/

public class Server {

ServerSocket socket;

Calculator calculator;

DataInputStream dis;

DataOutputStream dos;

NetworkManager networkManager;

public Server(){

calculator = new Calculator();

System.out.println(calculator.evaluate("5+6"));

start();

}

public void start(){

try {

socket = new ServerSocket(8192);

} catch (IOException e) {

e.printStackTrace();

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

}

Socket s;

try{

s = socket.accept();

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

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

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

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

String query = "";

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

query = networkManager.receive();

System.out.println("Inside query : " + query);

double answer = calculator.evaluate(query);

networkManager.send(answer + "");

}

} catch (IOException e) {

e.printStackTrace();

}

}

public static void main(String[] args){

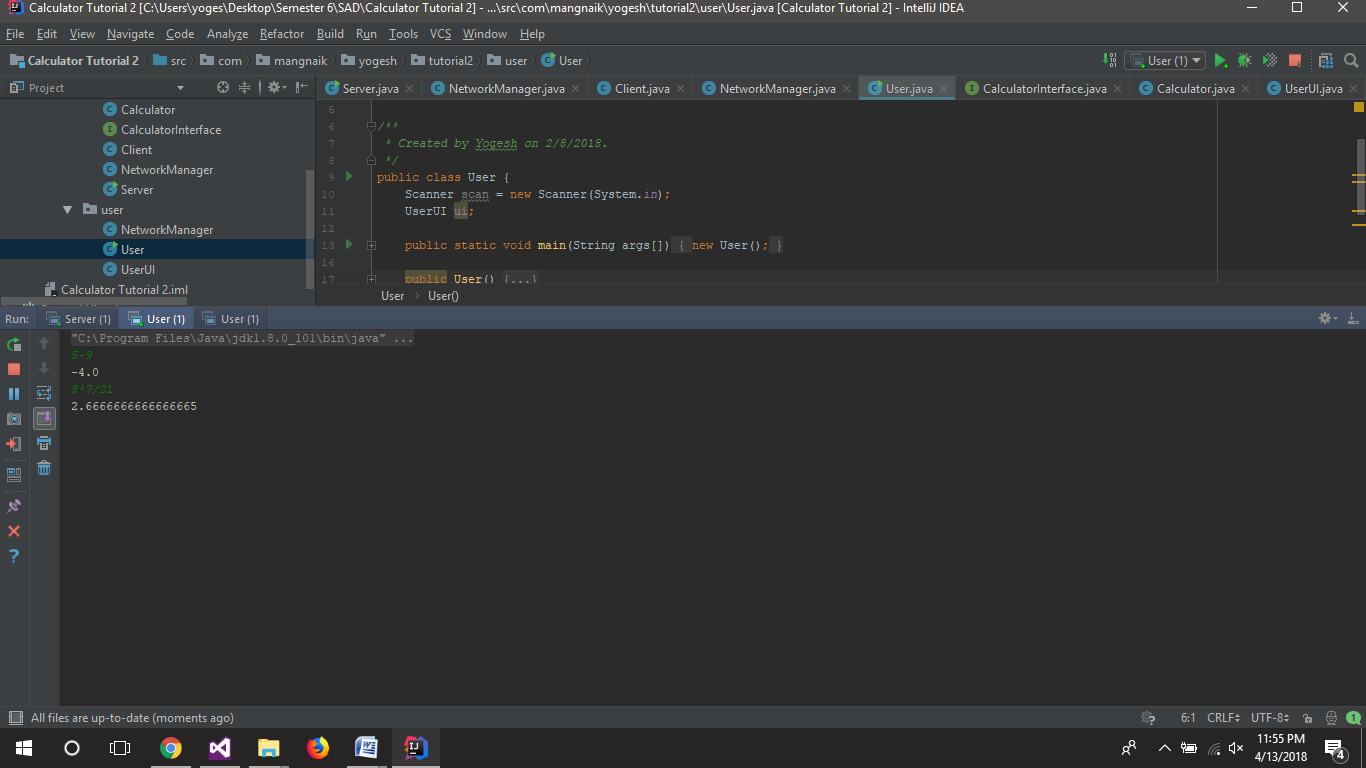
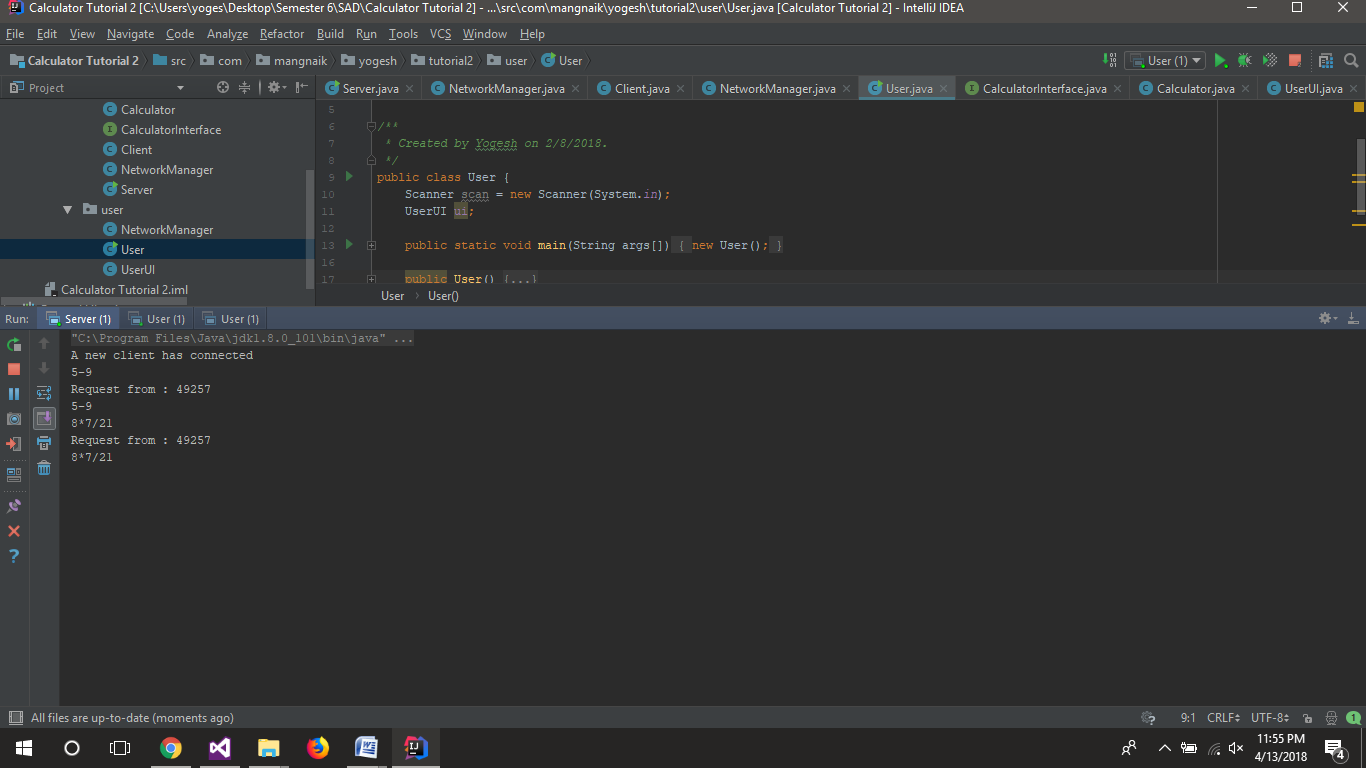
new Server();

}

}

**Output**

Client Server :



**Observation :**

When using monolithic architecture the code gets very big in one file and it gets difficult to manage and maintain the code. Also the reusability and extensibility of code is not very good. Monolithic architecture is good for small projects where size of code is very small. But it is not good for big projects as the files get very large and it gets increasingly difficult to maintain the code.

In server client architecture, we have only one thread. Therefore our server can serve only one client at this time. But if we add more threads we can server multiple clients at the same time. We have done this in the next tutorial.

**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();

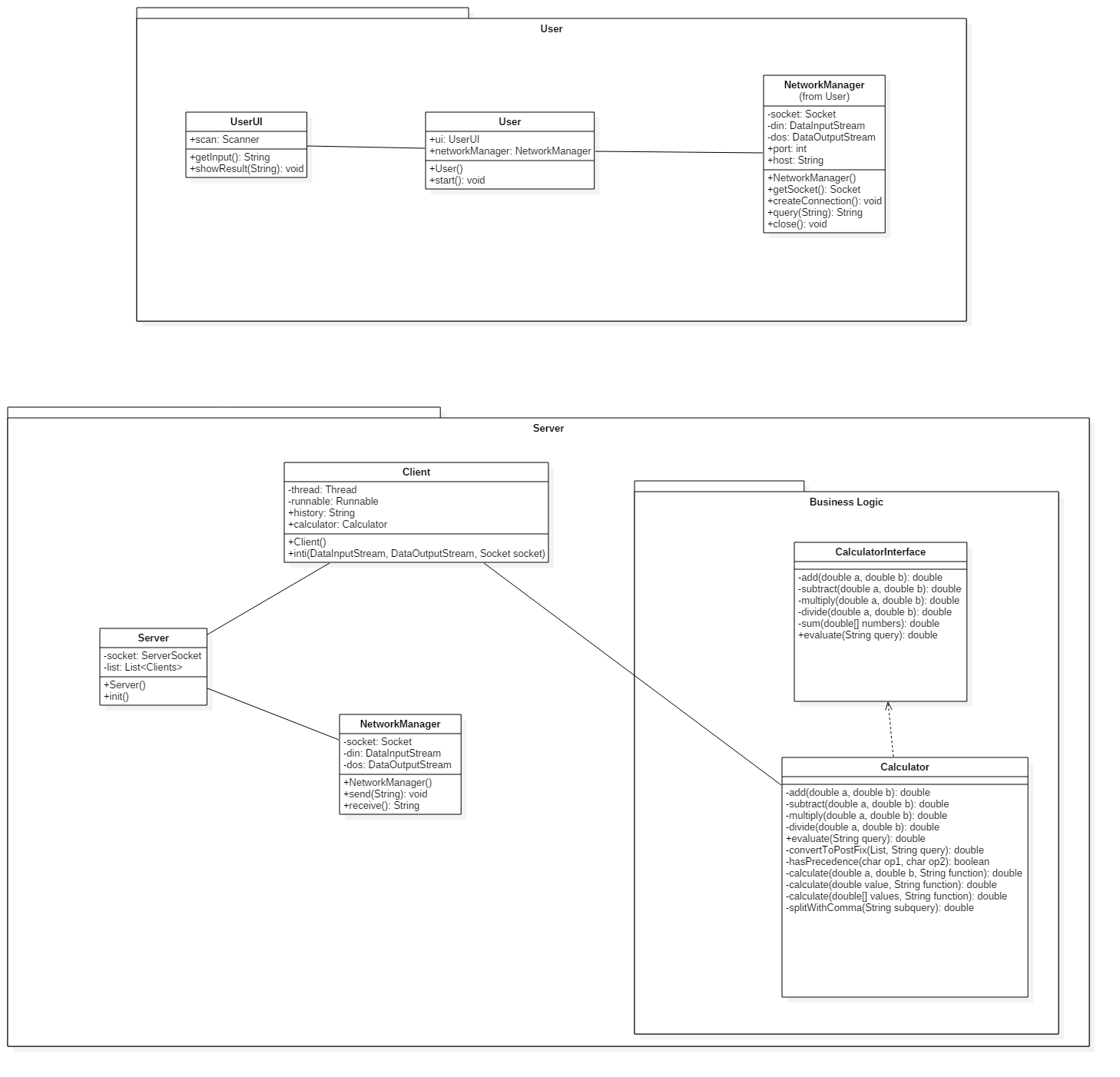
            object.start();

        }

    }

}

**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;

}

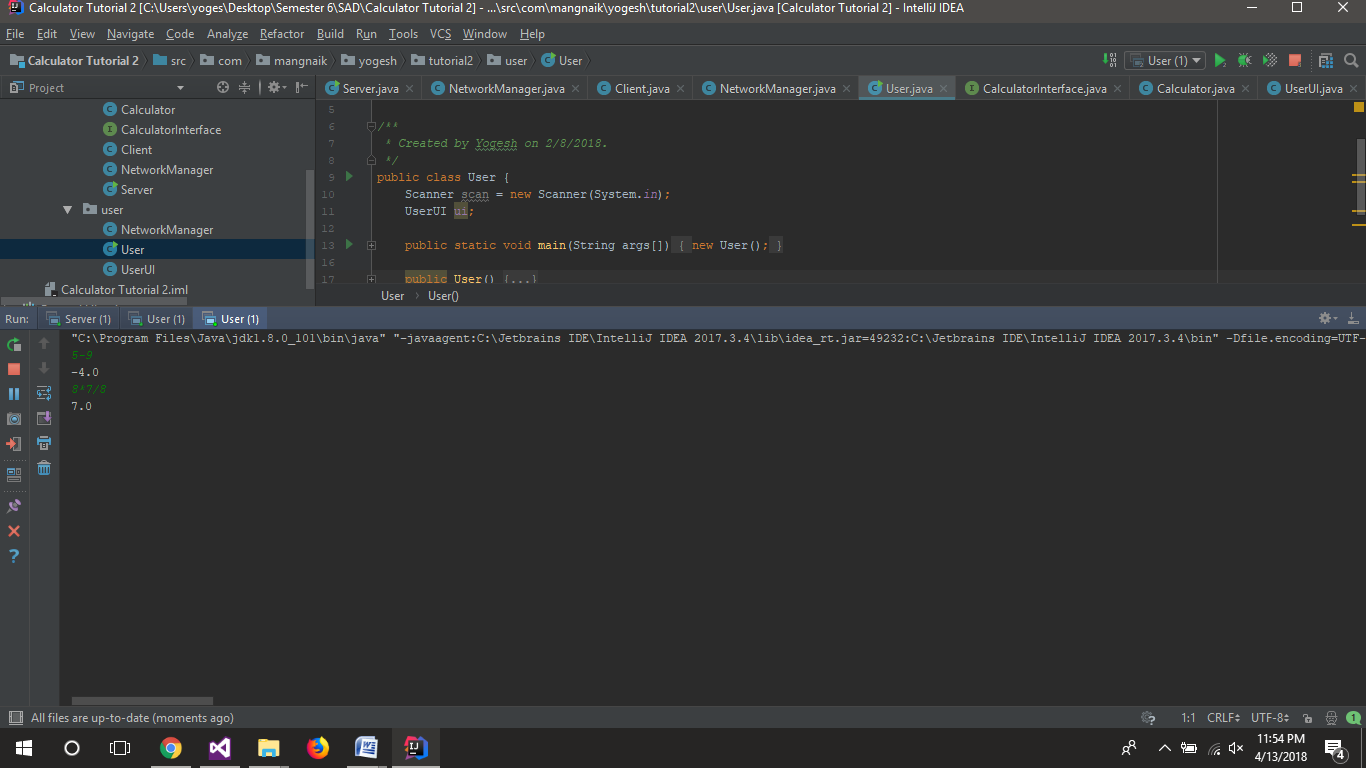
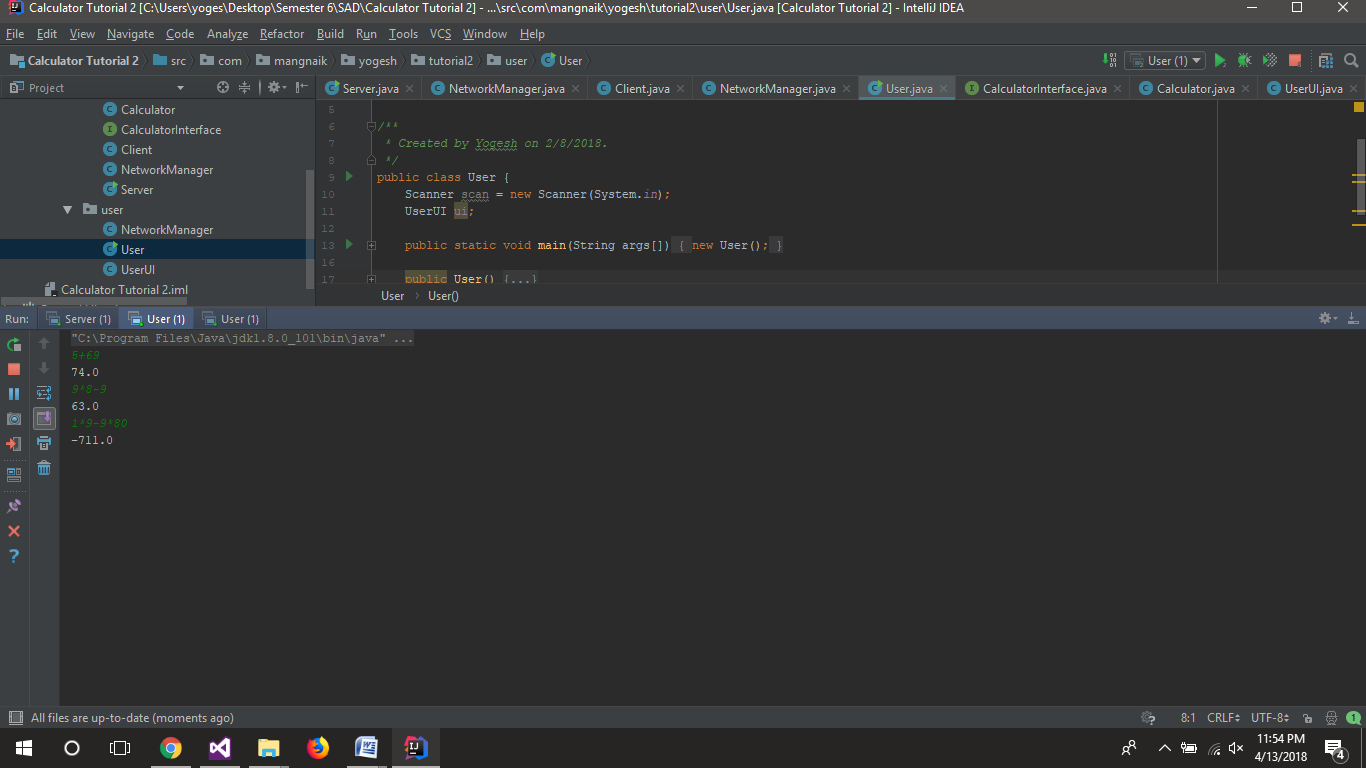
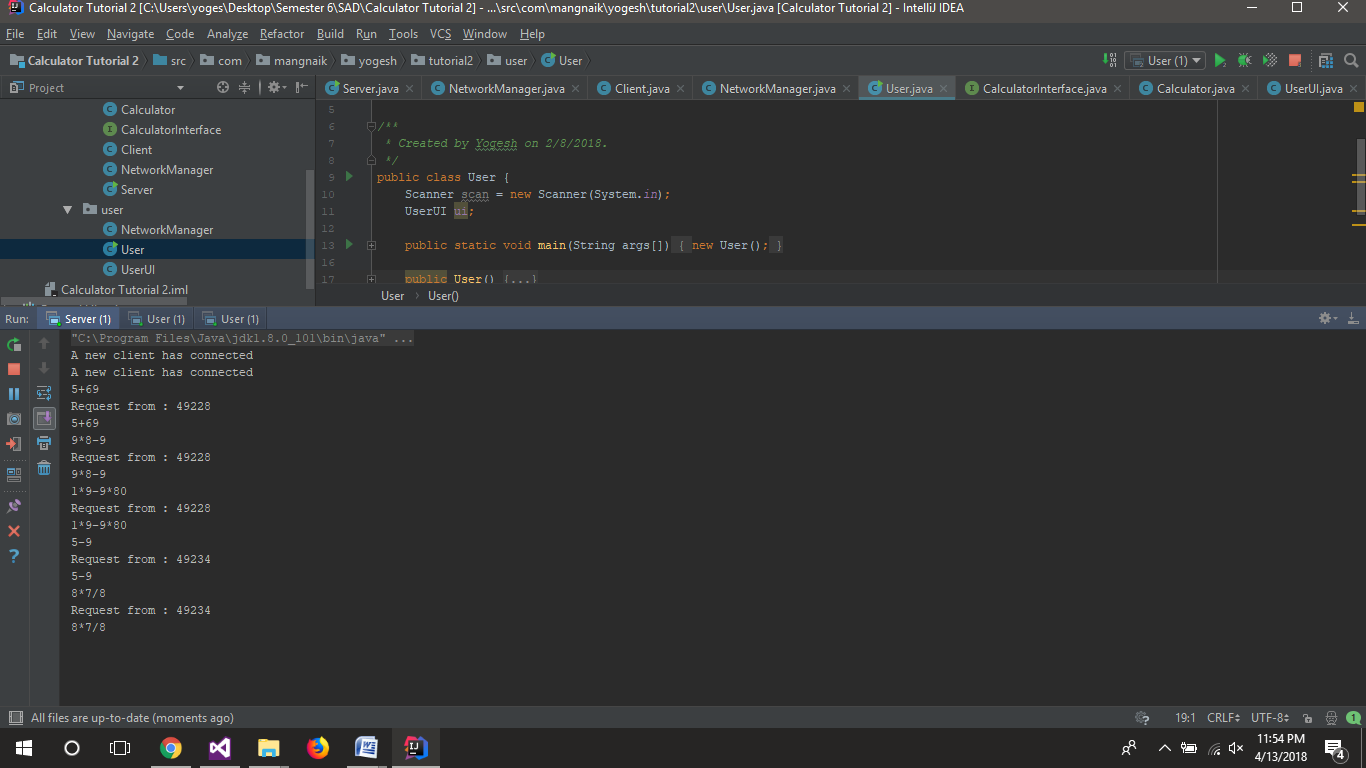
public void showResult(String answer){

System.out.println(answer);

}

}

**Output:**



**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.

**Tutorial No. 3**

**Problem statement:**

Extend Calculator application for :

1. Geometric Calculator
2. Statistical Calculator
3. Matrix Calculator

Apply Open Close Principle and use Factory method design pattern.

**Design Assumptions:**

In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), the open/closed principle states "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification" that is, such an entity can allow its behaviour to be extended without modifying its [source code](https://en.wikipedia.org/wiki/Source_code).

[Bertrand Meyer](https://en.wikipedia.org/wiki/Bertrand_Meyer) is generally credited for having originated the term open/closed principle, which appeared in his 1988 book [Object Oriented Software Construction](https://en.wikipedia.org/wiki/Object-Oriented_Software_Construction).

* A module will be said to be open if it is still available for extension. For example, it should be possible to add fields to the data structures it contains, or new elements to the set of functions it performs.
* A module will be said to be closed if [it] is available for use by other modules. This assumes that the module has been given a well-defined, stable description (the interface in the sense of information hiding).

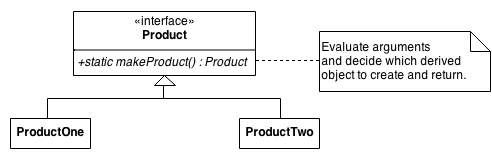
**Factory Design Pattern**

Factory Method is to creating objects as Template Method is to implementing an algorithm. A superclass specifies all standard and generic behavior (using pure virtual "placeholders" for creation steps), and then delegates the creation details to subclasses that are supplied by the client. Factory Method makes a design more customizable and only a little more complicated. Other design patterns require new classes, whereas Factory Method only requires a new operation.

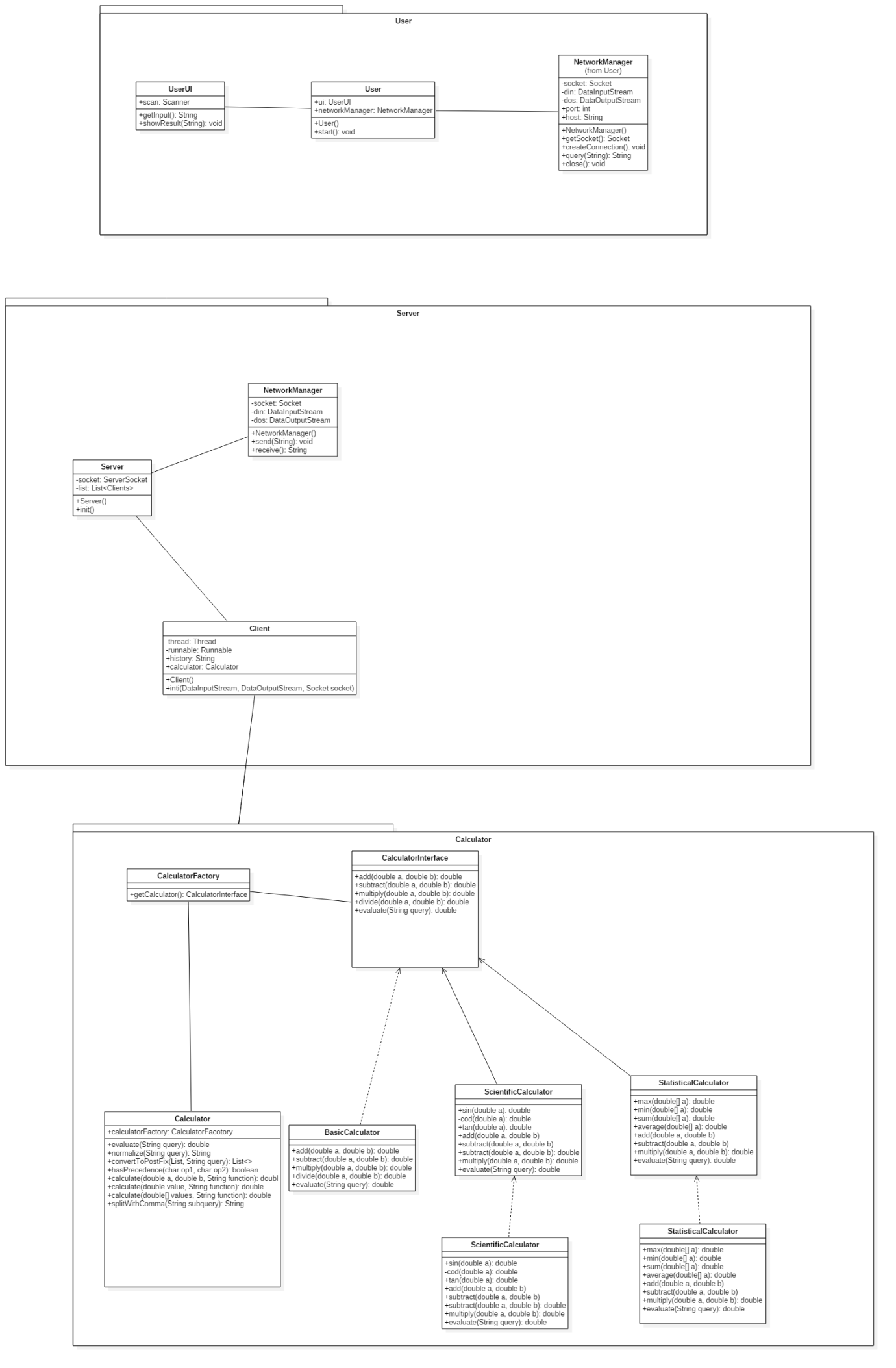
People often use Factory Method as the standard way to create objects; but it isn't necessary if: the class that's instantiated never changes, or instantiation takes place in an operation that subclasses can easily override (such as an initialization operation).

Factory Method is similar to Abstract Factory but without the emphasis on families.

Factory Methods are routinely specified by an architectural framework, and then implemented by the user of the framework.

[](https://en.wikipedia.org/wiki/Open/closed_principle#cite_note-2)

**Design Diagrams:**

****

**Code:**

**server side**

**ClientHandler.java**

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

package com.mangnaik.yogesh.server;

import com.mangnaik.yogesh.calculator.Calculator;

import com.mangnaik.yogesh.networkmanager.NetworkManagerServer;

import java.net.Socket;

/\*\*

\* Created by Yogesh on 2/2/2018.

\*/

class ClientHandler {

private Thread thread;

private Runnable runnable;

private String history;

private Calculator calculator;

ClientHandler(Socket socket) {

init(socket);

}

private void init(Socket socket){

NetworkManagerServer networkManager = new NetworkManagerServer(socket);

calculator = new Calculator();

history = "";

runnable = () -> {

while (true) {

System.out.println("ClientHandler is receiving");

String query = networkManager.listen();

System.out.println(query);

double answer = calculator.evaluate(query);

networkManager.send(answer+"");

}

};

thread = new Thread(runnable);

thread.start();

}

**Server.java**

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

package com.mangnaik.yogesh.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;

import com.mangnaik.yogesh.calculator.Calculator;

/\*\*

\* Created by Yogesh on 2/2/2018.

\*/

public class Server {

private ServerSocket socket;

private List<ClientHandler> clients = new ArrayList<>();

private Server(){

Calculator calculator = new Calculator();

init();

}

private 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");

clients.add(new ClientHandler(s));

} catch (IOException e) {

e.printStackTrace();

break;

}

}

}

public static void main(String[] args){

new Server();

}

}

**User.java**

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

package com.mangnaik.yogesh.user;

import com.mangnaik.yogesh.networkmanager.NetworkManagerClient;

/\*\*

\* Created by Yogesh on 2/2/2018.

\*/

public class User {

private UserUI ui;

private NetworkManagerClient networkManager;

public static void main(String args[]) {

new User();

}

private User() {

ui = new UserUI();

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

init();

}

private void init(){

networkManager.createConnection();

String query = "";

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

query = ui.getInput();

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

String ans;

ans = networkManager.send(query);

ui.showResult(ans);

}

}

}

}

**UserUI.java**

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

package com.mangnaik.yogesh.user;

import java.util.Scanner;

/\*\*

\* Created by Yogesh on 2/15/2018.

\*/

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);

}

}

**Calculator package**

**BasicCalculator.java**

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

package com.mangnaik.yogesh.calculator;

/\*\*

\* Created by Yogesh on 2/10/2018.

\*/

public class BasicCalculator implements BasicCalculatorInterface{

@Override

public double add(double a, double b) {

return a+b;

}

@Override

public double subtract(double a, double b) {

return b-a;

}

@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 calculate(double[] values, String function) {

switch(function){

case "+":

return add(values[0], values[1]);

case "-":

return subtract(values[1], values[0]);

case "\*":

return multiply(values[0], values[1]);

case "/":

return divide(values[1], values[0]);

}

return 0;

}

}

**BasicCalculatorInterface.java**

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

package com.mangnaik.yogesh.calculator;

/\*\*

\* Created by Yogesh on 2/2/2018.

\*/

public interface BasicCalculatorInterface {

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 calculate(double[] values, String function);

}

**Calculator.java**

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

package com.mangnaik.yogesh.calculator;

import com.sun.istack.internal.NotNull;

import java.util.ArrayList;

import java.util.List;

import java.util.Stack;

/\*\*

\* Created by Yogesh on 2/7/2018.

\*/

public class Calculator{

//call to evaluate the string query

public double evaluate(String query) {

query = query.replaceAll("pi", Math.PI+"");

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

query = normalize(query);

convertToPostFix(list, query);

return evaluate(list);

}

//evaluate the postfixed expression stored in list

private double evaluate(@NotNull List<String> list){

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

for (String aList : list) {

if (aList.equals("+") || aList.equals("-") || aList.equals("\*") || aList.equals("/")) {

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

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

double[] values = new double[]{a, b};

stack.push("" + calculate(values, aList.charAt(0) + ""));

} else {

stack.push(aList);

}

}

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

}

//evaluate and replace all the trigonometric and statistical terms

private String normalize(String query){

String[] functions = new String[]{"sin", "cos", "tan", "atan", "asin", "acos", "log"};

for (String function : functions) {

while (query.contains(function)) {

int index = query.indexOf(function);

int start = index + function.length();

int end = start;

int count = 0;

for (int j = start; j < query.length(); j++) {

end++;

if (query.charAt(j) == '(') {

count++;

} else if (query.charAt(j) == ')') {

count--;

if (count == 0) {

break;

}

}

}

String subquery = query.substring(start, end);

query = query.substring(0, index) + calculate(new double[]{evaluate(subquery)}, function) + query.substring(end, query.length());

}

}

functions = new String[]{"min", "max", "avg", "sum"};

for (String function : functions) {

while (query.contains(function)) {

int index = query.indexOf(function);

int start = index + 3;

int end = start;

int count = 0;

for (int j = start; j < query.length(); j++) {

end++;

if (query.charAt(j) == '{') {

count++;

} else if (query.charAt(j) == '}') {

count--;

if (count == 0) {

break;

}

}

}

String subquery = query.substring(start + 1, end - 1);

double[] values = splitWithComma(subquery);

query = query.substring(0, index) + calculate(values, function) + query.substring(end, query.length());

}

}

return query;

}

//convert the string to postfix and store the result in a list

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()+"");

}

//check for precedence

private boolean hasPrecedence(char op1, char op2) {

return op2 != '(' && op2 != ')' && ((op1 != '\*' && op1 != '/') || (op2 != '+' && op2 != '-'));

}

//call the evalutate function of calculators

private double calculate(double[] values, String function){

BasicCalculatorInterface calculator = CalculatorFactory.getCalculator(function);

return calculator.calculate(values, function);

}

//splitting with comma

private double[] splitWithComma(String subquery){

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

int bracketCount = 0;

int s = 0;

for(int k=0; k<subquery.length(); k++){

if(subquery.charAt(k)=='{'){

bracketCount++;

}

else if(subquery.charAt(k)=='}'){

bracketCount--;

}

if(bracketCount==0){

if(subquery.charAt(k)==','){

String st = subquery.substring(s,k);

elementList.add(st);

s=k+1;

}

}

}

if(!subquery.substring(s,subquery.length()).equals("")){

String st = subquery.substring(s, subquery.length());

elementList.add(st);

}

String[] elements = new String[elementList.size()];

for(int k=0; k<elementList.size(); k++){

elements[k] = elementList.get(k);

}

double[] values = new double[elements.length];

for(int j=0; j<values.length; j++){

values[j] = evaluate(elements[j]);

}

return values;

}

}

**CalculatorFactory.java**

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

package com.mangnaik.yogesh.calculator;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Created by Yogesh on 2/7/2018.

\*/

public class CalculatorFactory {

private static List<String> scientificFunctions;

private static List<String> basicFunctions;

private static List<String> statisticalFunctions;

static{

scientificFunctions = new ArrayList<>();

basicFunctions = new ArrayList<>();

statisticalFunctions = new ArrayList<>();

scientificFunctions.add("sin");

scientificFunctions.add("cos");

scientificFunctions.add("tan");

basicFunctions.add("+");

basicFunctions.add("-");

basicFunctions.add("\*");

basicFunctions.add("/");

statisticalFunctions.add("max");

statisticalFunctions.add("min");

statisticalFunctions.add("avg");

statisticalFunctions.add("sum");

}

static BasicCalculatorInterface getCalculator(String function){

BasicCalculatorInterface basicCalculatorInterface = null;

if(scientificFunctions.contains(function)){

basicCalculatorInterface = new ScientificCalculator();

}

else if(basicFunctions.contains(function)){

basicCalculatorInterface = new BasicCalculator();

}

else if(statisticalFunctions.contains(function)){

basicCalculatorInterface = new StatisticalCalculator();

}

return basicCalculatorInterface;

}

}

**ScientificCalculator.java**

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

package com.mangnaik.yogesh.calculator;

/\*\*

\* Created by Yogesh on 2/3/2018.

\*/

public class ScientificCalculator implements ScientificCalculatorInterface{

@Override

public double sin(double angle) {

System.out.println("sin + " + angle);

System.out.println(Math.sin(angle));

return Math.sin(angle);

}

@Override

public double cos(double angle) {

return Math.cos(angle);

}

@Override

public double tan(double angle) {

return Math.tan(angle);

}

@Override

public double asin(double value) {

return Math.asin(value);

}

@Override

public double acos(double value) {

return Math.acos(value);

}

@Override

public double atan(double value) {

return Math.atan(value);

}

@Override

public double log(double number) {

return Math.log(number);

}

@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 calculate(double[] values, String function) {

switch(function){

case "sin":

return sin(values[0]);

case "cos":

return cos(values[0]);

case "tan":

return tan(values[0]);

case "asin":

return asin(values[0]);

case "acos":

return acos(values[0]);

case "atan":

return atan(values[0]);

case "log":

return log(values[0]);

}

return 0;

}

}

**ScientificCalculatorInterface.java**

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

package com.mangnaik.yogesh.calculator;

public interface ScientificCalculatorInterface extends BasicCalculatorInterface {

//Trigonometric

public double sin(double angle);

public double cos(double angle);

public double tan(double angle);

}

**StatisticalCalculator.java**

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

package com.mangnaik.yogesh.calculator;

/\*\*

\* Created by Yogesh on 2/7/2018.

\*/

public class StatisticalCalculator implements StatisticalCalculatorInterface{

@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 average(double[] values) {

double sum = 0;

for (double number : values) {

sum += number;

}

return sum/values.length;

}

@Override

public double max(double[] values) {

double max = -Integer.MAX\_VALUE;

for (double number : values) {

if (number > max) {

max = number;

}

}

return max;

}

@Override

public double min(double[] values) {

double min = Integer.MAX\_VALUE;

for (double number : values) {

if (number < min) {

min = number;

}

}

return min;

}

@Override

public double sum(double[] values) {

double sum = 0;

for (double number : values) {

sum += number;

}

return sum;

}

@Override

public double calculate(double[] values, String function) {

switch (function){

case "max":

return max(values);

case "min":

return min(values);

case "avg":

return average(values);

case "sum":

return sum(values);

}

return 0;

}

}

**StatisticalCalculatorInterface.java**

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

package com.mangnaik.yogesh.calculator;

public interface StatisticalCalculatorInterface extends BasicCalculatorInterface {

public double average(double[] numbers);

public double max(double[] numbers);

public double min(double[] numbers);

public double sum(double[] numbers);

}

**NetworkManager package**

**NetworkManagerClient.java**

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

package com.mangnaik.yogesh.networkmanager;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class NetworkManagerClient {

private Socket socket;

private DataInputStream dis;

private DataOutputStream dos;

private int port;

private String host;

public NetworkManagerClient(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();

return;

}

try {

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

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

} catch (IOException e) {

e.printStackTrace();

}

}

public String send(String query) {

String answer = "";

try {

dos.writeUTF(query);

answer = dis.readUTF();

} catch (IOException e) {

close();

e.printStackTrace();

}

return answer;

}

private void close() {

try {

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**NetworkManagerServer.java**

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

package com.mangnaik.yogesh.networkmanager;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class NetworkManagerServer {

private DataInputStream dis;

private DataOutputStream dos;

private Socket socket;

public NetworkManagerServer(Socket socket) {

this.socket = socket;

try {

this.dis = new DataInputStream(socket.getInputStream());

this.dos = new DataOutputStream(socket.getOutputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

public String listen(){

String received;

try {

received = dis.readUTF();

return received;

} catch (IOException e) {

close();

e.printStackTrace();

}

return "";

}

public void send(String answer){

try{

dos.writeUTF(answer);

}

catch (Exception e){

e.printStackTrace();

}

}

private void close() {

try {

socket.close();

} catch (IOException e) {

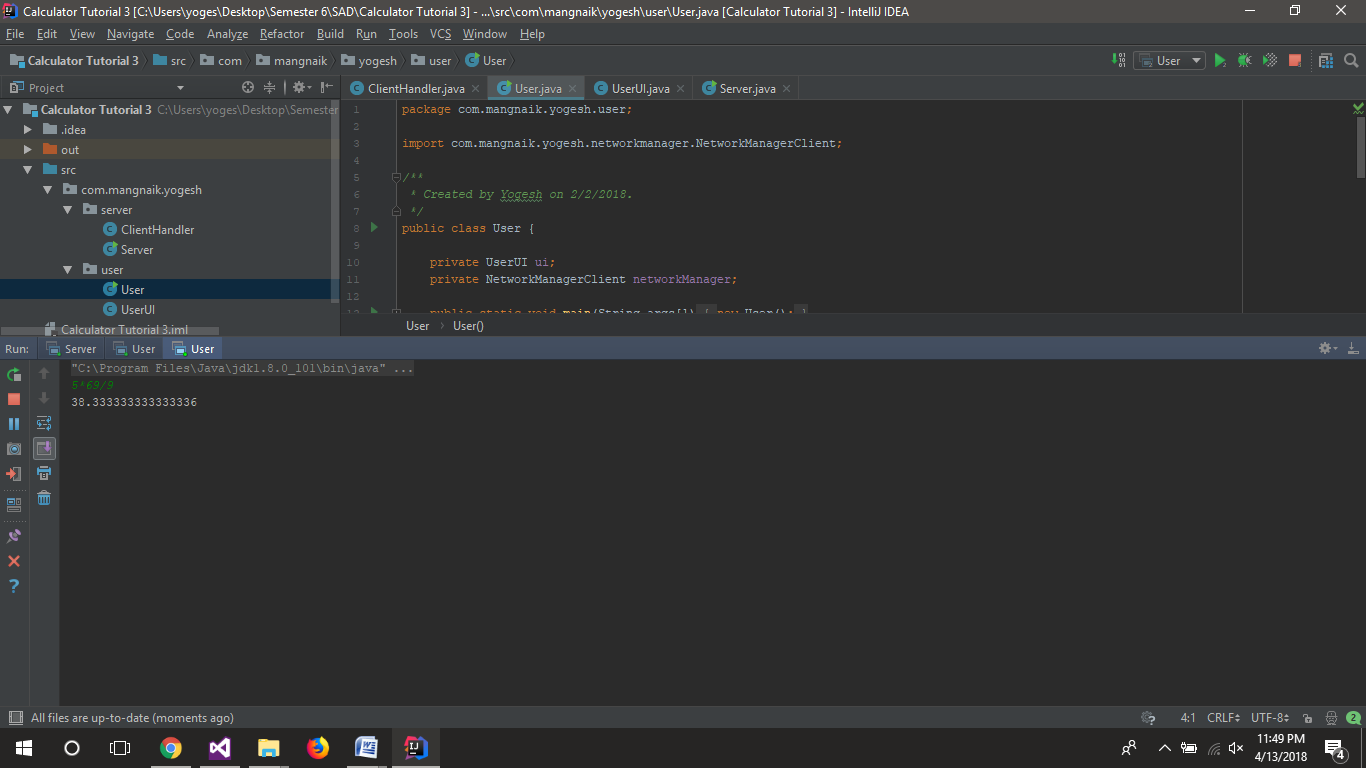
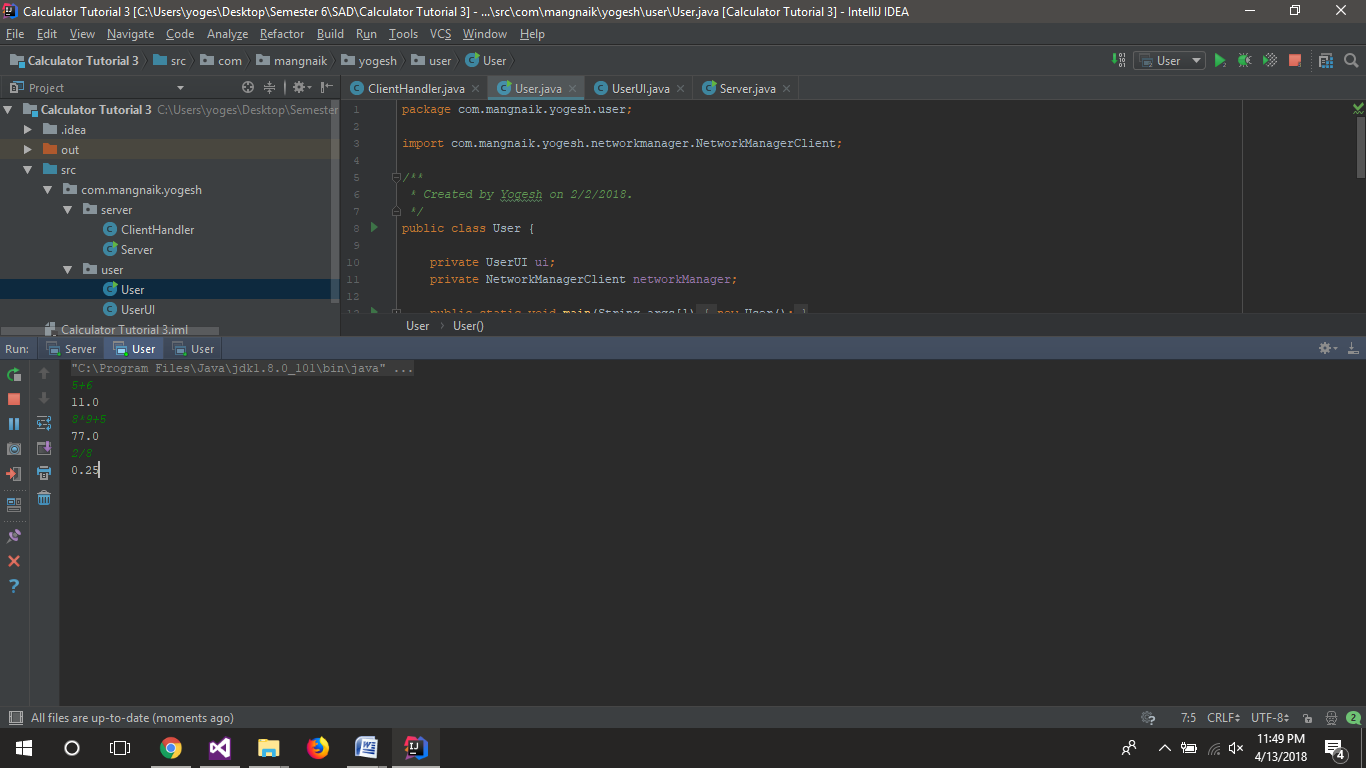
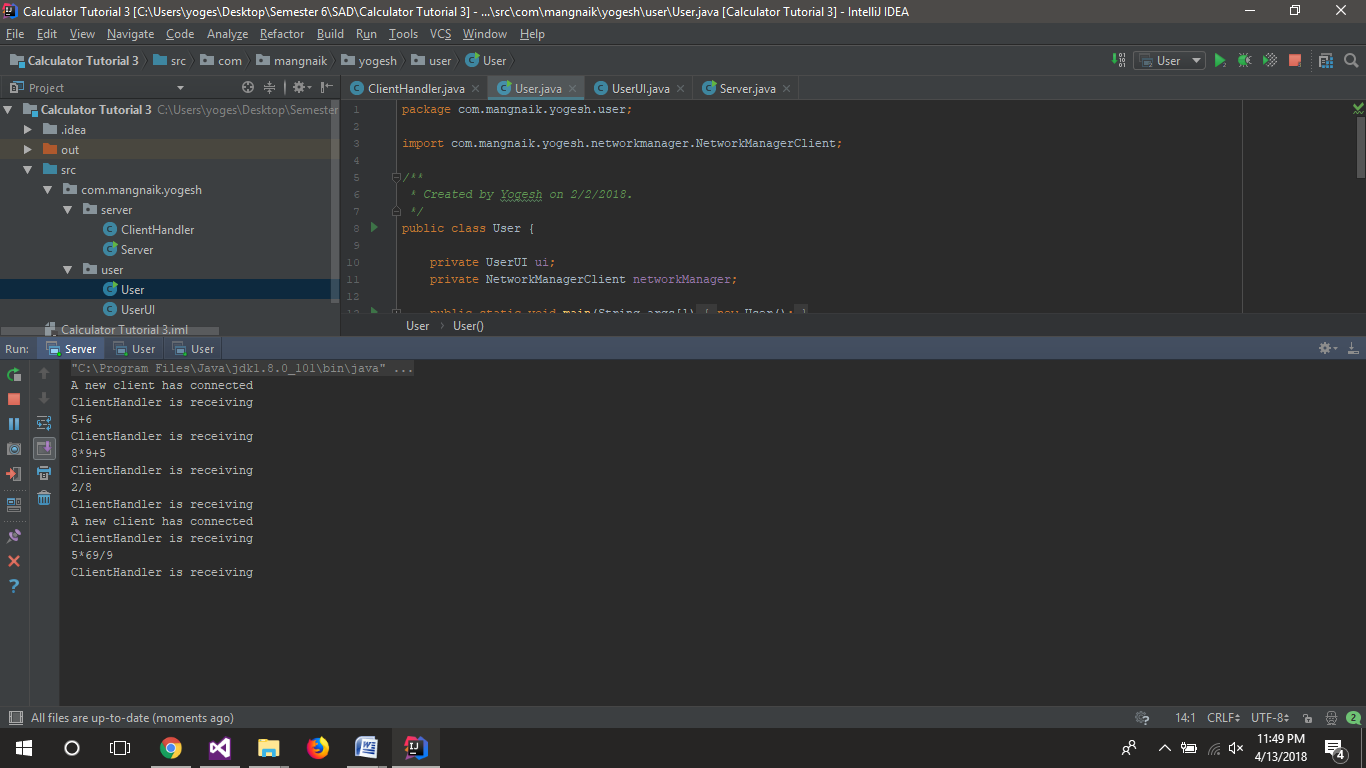
e.printStackTrace();

}

}

}

**Output:**



**Observation:**

While implementing open close principle we have to make a base class which will be an interface and declare all the methods which will be used by the classes that implement this interface. While designing this interface care has to be taken to not violate other principles such as including unnecessary functions in the interface.

While designing the factory method of producing objects we have to consider which type of input we should get to decide which object we should produce. Here we have taken string which tells the factory which calculator the user needs. We can also use constants defined in other class to tell which object is needed at the runtime. Factory pattern is very useful for extensibility and runtime binding of objects.

**Tutorial No. 4**

**Problem statement:**

Extend Calculator program to implement MVC architecture. Design JFrame to create view, use basic calculator as model and create appropriate controller.

**Design Assumptions:**

Model–view–controller (MVC) is an [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) commonly used for developing [user interfaces](https://en.wikipedia.org/wiki/User_interface) that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development.

Traditionally used for desktop [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUIs), this architecture has become popular for designing [web applications](https://en.wikipedia.org/wiki/Web_application) and even mobile, desktop and other clients. Popular programming languages like [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [PHP](https://en.wikipedia.org/wiki/PHP) and others have popular MVC frameworks that are currently being used in web application development straight [out of the box](https://en.wikipedia.org/wiki/Out_of_the_box_(feature)).

**Components**

The model is the central component of the pattern. It expresses the application's behavior in terms of the [problem domain](https://en.wikipedia.org/wiki/Problem_domain), independent of the user interface. It directly manages the data, logic and rules of the application.

* A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
* The third part or section, the controller, accepts input and converts it to commands for the model or view.

**Interactions**

In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.

* The model is responsible for managing the data of the application. It receives user input from the controller.
* The view means presentation of the model in a particular format.
* The controller is responsible for responding to the user input and perform interactions on the data model objects. The controller receives the input, optionally validates the input and then passes the input to the model.

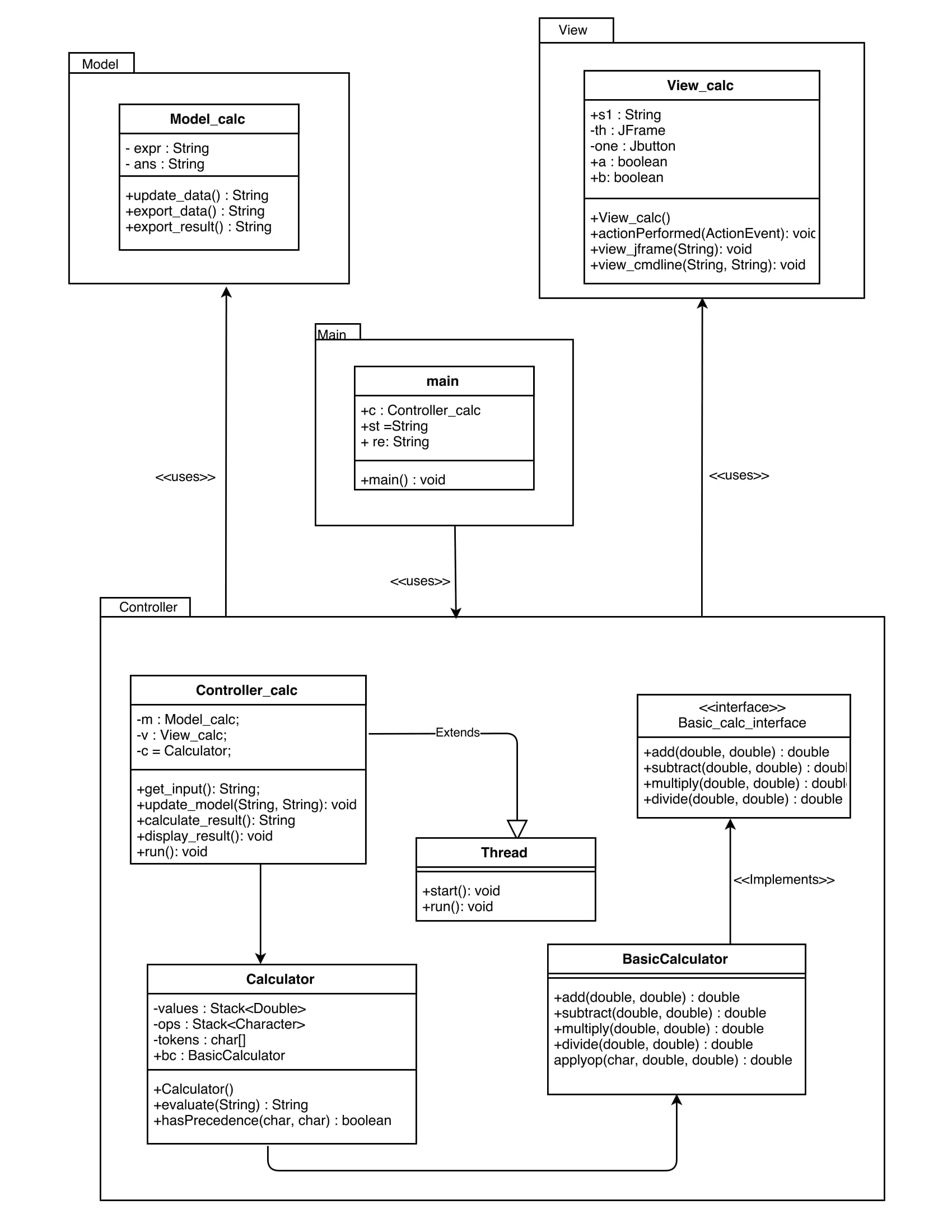
**Advantages**

* Simultaneous development – Multiple developers can work simultaneously on the model, controller and views.
* [High cohesion](https://en.wikipedia.org/wiki/Cohesion_(computer_science)) – MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped together.
* [Low coupling](https://en.wikipedia.org/wiki/Loose_coupling) – The very nature of the MVC framework is such that there is low coupling among models, views or controllers
* Ease of modification – Because of the separation of responsibilities, future development or modification is easier
* Multiple views for a model – Models can have multiple views

**Disadvantages**

* Code navigability – The framework navigation can be complex because it introduces new layers of abstraction and requires users to adapt to the decomposition criteria of MVC.
* Multi-artifact consistency – Decomposing a feature into three artifacts causes scattering. Thus, requiring developers to maintain the consistency of multiple representations at once.
* Pronounced learning curve – Knowledge on multiple technologies becomes the norm. Developers using MVC need to be skilled in multiple technologies.

**Design Diagrams:**

****

**Code:**

**Controller\_calc.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Controller;

import Model.Model\_calc;

import View.View\_calc;

import calculator.\*;

import java.util.Scanner;

/\*

\* @author Jatin

\*/

public class Controller\_calc extends Thread {

private Model\_calc m;

private View\_calc v;

private Calculator c;

private volatile String c\_input = "";

private Scanner scn = new Scanner(System.in);

public Controller\_calc(Model\_calc m, View\_calc v){

this.m = m;

this.v = v;

c = new Calculator();

}

public String get\_input(){

while(v.a!=true)

{

if(!"".equals(c\_input)){

String to\_send = c\_input;

c\_input="";

return to\_send;

}

}

if(v.a==true){

v.a = false;

return v.s1;

}

return null; //will never execute

}

public void update\_model(String st, String ans){

m.update\_data(st,ans);

}

public String calculate\_result(){

return c.evaluate(m.export\_data());

}

public void display\_result(){

v.view\_jframe(m.export\_result());

v.view\_cmdline(m.export\_data(),m.export\_result());

}

@Override

public void run() {

c\_input = scn.next();

v.b=true;

}

}

**main.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Main;

import Model.Model\_calc;

import View.View\_calc;

import Controller.Controller\_calc;

/\*\*

\*

\* @author Jatin

\*/

public class Main {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Controller\_calc c = new Controller\_calc(new Model\_calc(), new View\_calc());

c.start();

String st ="";

st = c.get\_input();

while(true){

String re = "0";

c.update\_model(st, re);

re = c.calculate\_result();

c.update\_model(st, re);

c.display\_result();

st = c.get\_input();

}

}

}

**Model\_calc.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Model;

/\*\*

\*

\* @author Jatin

\*/

public class Model\_calc {

private static String expr;

private static String ans;

public void update\_data(String st, String result){

expr = st;

ans = result;

}

public String export\_data(){

return expr;

}

public String export\_result(){

return ans;

}

}

**View\_calc.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package View;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

/\*\*

\*

\* @author Jatin

\*/

public class View\_calc implements ActionListener{

JTextField t;

JButton dec;

JButton equal;

JButton zero;

JButton one;

JButton two;

JButton three;

JButton four;

JButton five;

JButton six;

JButton seven;

JButton eight;

JButton nine;

JButton plus;

JButton minus;

JButton multiply;

JButton divide;

JButton clear;

private JFrame th;

public boolean a=false;

public boolean b=true;

public String s1;

public View\_calc() {

th = new JFrame("Basic Math calculator");

t = new JTextField();

t.setBounds(45,45,245,40);

one = new JButton("1");

one.setBounds(45,110,50,50);

four = new JButton("4");

four.setBounds(45,175,50,50);

seven = new JButton("7");

seven.setBounds(45,240,50,50);

dec = new JButton(".");

dec.setBounds(45,305,50,50);

two = new JButton("2");

two.setBounds(110,110,50,50);

five = new JButton("5");

five.setBounds(110,175,50,50);

eight = new JButton("8");

eight.setBounds(110,240,50,50);

zero = new JButton("0");

zero.setBounds(110,305,50,50);

three = new JButton("3");

three.setBounds(175,110,50,50);

six = new JButton("6");

six.setBounds(175,175,50,50);

nine = new JButton("9");

nine.setBounds(175,240,50,50);

equal = new JButton("=");

equal.setBounds(175,305,50,50);

plus = new JButton("+");

plus.setBounds(240,110,50,50);

minus = new JButton("-");

minus.setBounds(240,175,50,50);

multiply = new JButton("\*");

multiply.setBounds(240,240,50,50);

divide = new JButton("/");

divide.setBounds(240,305,50,50);

clear = new JButton("clear");

clear.setBounds(110,370,100,50);

th.setSize(350, 470);

th.add(one);

th.add(two);

th.add(three);

th.add(four);

th.add(five);

th.add(six);

th.add(seven);

th.add(eight);

th.add(nine);

th.add(dec);

th.add(zero);

th.add(equal);

th.add(plus);

th.add(multiply);

th.add(divide);

th.add(minus);

th.add(t);

th.add(clear);

th.setLayout(null);

th.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

th.setVisible(true);

one.addActionListener(this);

two.addActionListener(this);

three.addActionListener(this);

four.addActionListener(this);

five.addActionListener(this);

six.addActionListener(this);

seven.addActionListener(this);

eight.addActionListener(this);

nine.addActionListener(this);

zero.addActionListener(this);

equal.addActionListener(this);

plus.addActionListener(this);

multiply.addActionListener(this);

divide.addActionListener(this);

minus.addActionListener(this);

dec.addActionListener(this);

clear.addActionListener(this);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==one){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("1"));

}

if(e.getSource()==two){

{

if(this.b==true){

this.b=false;

t.setText("");

}t.setText(t.getText().concat("2"));

}

}

if(e.getSource()==three){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("3"));

}

if(e.getSource()==four){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("4"));

}

if(e.getSource()==five){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("5"));

}

if(e.getSource()==six){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("6"));

}

if(e.getSource()==seven){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("7"));

}

if(e.getSource()==eight){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("8"));

}

if(e.getSource()==nine){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("9"));

}

if(e.getSource()==zero){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("0"));

}

if(e.getSource()==dec){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("."));

}

if(e.getSource()==plus)

t.setText(t.getText().concat("+"));

if(e.getSource()==multiply)

t.setText(t.getText().concat("\*"));

if(e.getSource()==divide)

t.setText(t.getText().concat("/"));

if(e.getSource()==minus)

t.setText(t.getText().concat("-"));

if(e.getSource()==clear)

{

t.setText("");

a=false;

}

if(e.getSource()==equal)

{

this.s1=t.getText();

if("".equals(s1))

{

JOptionPane.showMessageDialog(th, "Please enter expression");

}

else {

this.a=true;

this.b=true;

}

}

}

public void view\_jframe(String ans){

t.setText(ans);

}

public void view\_cmdline(String exp, String ans){

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

}

}

**Output:**

**Observation:**

**Tutorial No. 5**

**Problem statement:**

Implement an application for the TPO at VJTI forplacements implementing various activites that place like internship, placement student info. etc.

**Design Assumptions:**

**Microservices Architecture**

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack.

The microservice architecture is not a silver bullet. It has several drawbacks. Moreover, when using this architecture there are numerous issues that you must address. The microservice architecture pattern language is a collection of patterns for applying the microservice architecture. It has two goals:

1. The pattern language enables you to decide whether microservices are a good fit for your application.
2. The pattern language enables you to use the microservice architecture successfully.

Flask is a micro [web framework](https://en.wikipedia.org/wiki/Web_framework) written in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) and based on the Werkzeug toolkit and [Jinja2](https://en.wikipedia.org/wiki/Jinja_(template_engine)) template engine. It is [BSD licensed](https://en.wikipedia.org/wiki/BSD_licenses).

The latest stable version of Flask is 0.12.2 as of May 2017. Applications that use the Flask framework include [Pinterest](https://en.wikipedia.org/wiki/Pinterest), [LinkedIn](https://en.wikipedia.org/wiki/LinkedIn), and the community web page for Flask itself.

Flask is called a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.[[7]](https://en.wikipedia.org/wiki/Flask_(web_framework)#cite_note-7)

The route() decorator in Flask is used to bind URL to a function. For example −

@app.route(‘/hello’)

def hello\_world():

return ‘hello world’

Here, URL ‘/hello’ rule is bound to the hello\_world() function. As a result, if a user visits http://localhost:5000/hello URL, the output of the hello\_world() function will be rendered in the browser.

The add\_url\_rule() function of an application object is also available to bind a URL with a function as in the above example, route() is used.

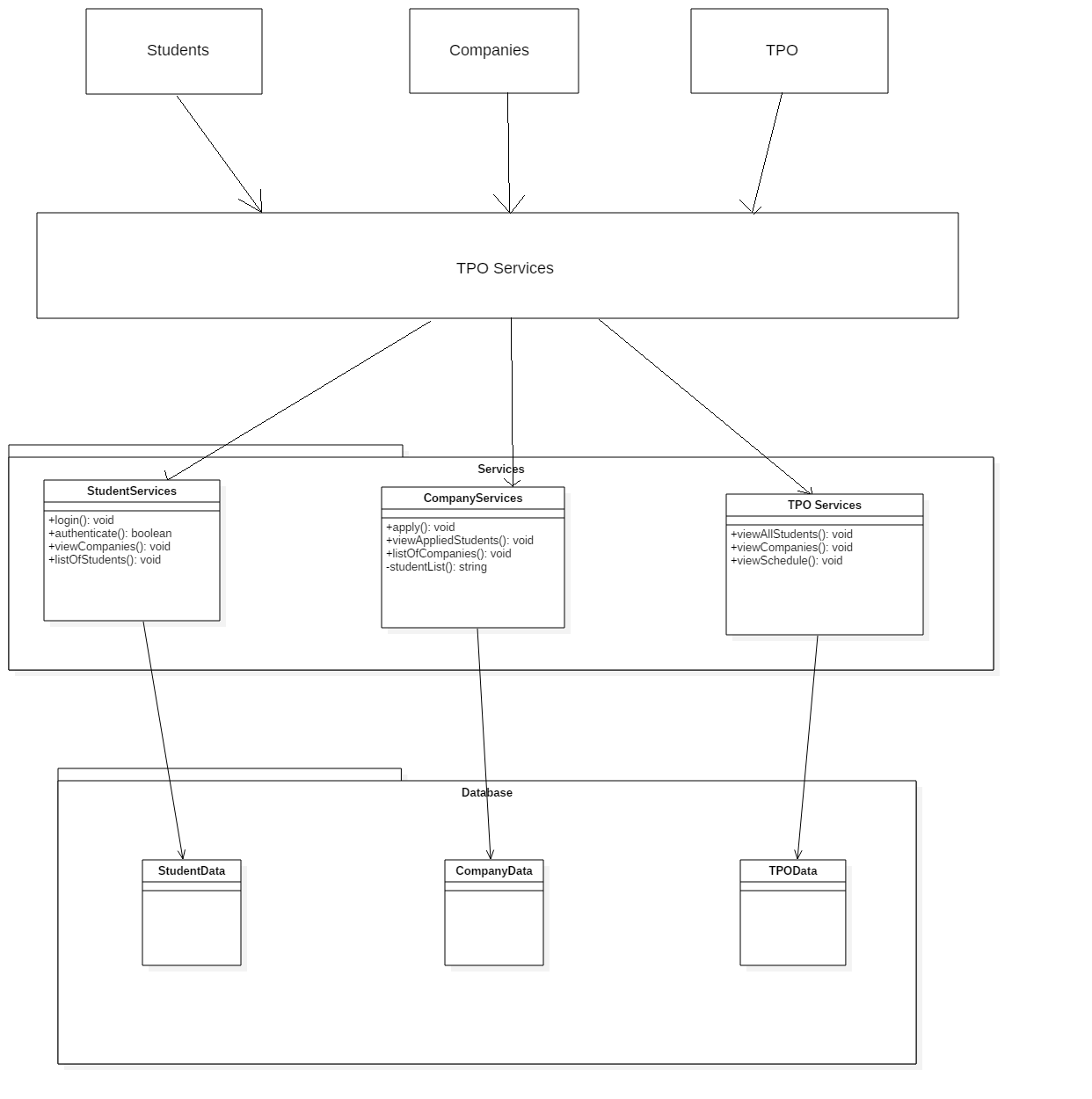
A decorator’s purpose is also served by the following representation −

def hello\_world():

return ‘hello world’

app.add\_url\_rule(‘/’, ‘hello’, hello\_world)

**Design Diagrams:**

****

**Code:**

**Controller\_calc.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Controller;

import Model.Model\_calc;

import View.View\_calc;

import calculator.\*;

import java.util.Scanner;

/\*

\* @author Jatin

\*/

public class Controller\_calc extends Thread {

private Model\_calc m;

private View\_calc v;

private Calculator c;

private volatile String c\_input = "";

private Scanner scn = new Scanner(System.in);

public Controller\_calc(Model\_calc m, View\_calc v){

this.m = m;

this.v = v;

c = new Calculator();

}

public String get\_input(){

while(v.a!=true)

{

if(!"".equals(c\_input)){

String to\_send = c\_input;

c\_input="";

return to\_send;

}

}

if(v.a==true){

v.a = false;

return v.s1;

}

return null; //will never execute

}

public void update\_model(String st, String ans){

m.update\_data(st,ans);

}

public String calculate\_result(){

return c.evaluate(m.export\_data());

}

public void display\_result(){

v.view\_jframe(m.export\_result());

v.view\_cmdline(m.export\_data(),m.export\_result());

}

@Override

public void run() {

c\_input = scn.next();

v.b=true;

}

}

**main.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Main;

import Model.Model\_calc;

import View.View\_calc;

import Controller.Controller\_calc;

/\*\*

\*

\* @author Jatin

\*/

public class Main {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Controller\_calc c = new Controller\_calc(new Model\_calc(), new View\_calc());

c.start();

String st ="";

st = c.get\_input();

while(true){

String re = "0";

c.update\_model(st, re);

re = c.calculate\_result();

c.update\_model(st, re);

c.display\_result();

st = c.get\_input();

}

}

}

**Model\_calc.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Model;

/\*\*

\*

\* @author Jatin

\*/

public class Model\_calc {

private static String expr;

private static String ans;

public void update\_data(String st, String result){

expr = st;

ans = result;

}

public String export\_data(){

return expr;

}

public String export\_result(){

return ans;

}

}

**View\_calc.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package View;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

/\*\*

\*

\* @author Jatin

\*/

public class View\_calc implements ActionListener{

JTextField t;

JButton dec;

JButton equal;

JButton zero;

JButton one;

JButton two;

JButton three;

JButton four;

JButton five;

JButton six;

JButton seven;

JButton eight;

JButton nine;

JButton plus;

JButton minus;

JButton multiply;

JButton divide;

JButton clear;

private JFrame th;

public boolean a=false;

public boolean b=true;

public String s1;

public View\_calc() {

th = new JFrame("Basic Math calculator");

t = new JTextField();

t.setBounds(45,45,245,40);

one = new JButton("1");

one.setBounds(45,110,50,50);

four = new JButton("4");

four.setBounds(45,175,50,50);

seven = new JButton("7");

seven.setBounds(45,240,50,50);

dec = new JButton(".");

dec.setBounds(45,305,50,50);

two = new JButton("2");

two.setBounds(110,110,50,50);

five = new JButton("5");

five.setBounds(110,175,50,50);

eight = new JButton("8");

eight.setBounds(110,240,50,50);

zero = new JButton("0");

zero.setBounds(110,305,50,50);

three = new JButton("3");

three.setBounds(175,110,50,50);

six = new JButton("6");

six.setBounds(175,175,50,50);

nine = new JButton("9");

nine.setBounds(175,240,50,50);

equal = new JButton("=");

equal.setBounds(175,305,50,50);

plus = new JButton("+");

plus.setBounds(240,110,50,50);

minus = new JButton("-");

minus.setBounds(240,175,50,50);

multiply = new JButton("\*");

multiply.setBounds(240,240,50,50);

divide = new JButton("/");

divide.setBounds(240,305,50,50);

clear = new JButton("clear");

clear.setBounds(110,370,100,50);

th.setSize(350, 470);

th.add(one);

th.add(two);

th.add(three);

th.add(four);

th.add(five);

th.add(six);

th.add(seven);

th.add(eight);

th.add(nine);

th.add(dec);

th.add(zero);

th.add(equal);

th.add(plus);

th.add(multiply);

th.add(divide);

th.add(minus);

th.add(t);

th.add(clear);

th.setLayout(null);

th.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

th.setVisible(true);

one.addActionListener(this);

two.addActionListener(this);

three.addActionListener(this);

four.addActionListener(this);

five.addActionListener(this);

six.addActionListener(this);

seven.addActionListener(this);

eight.addActionListener(this);

nine.addActionListener(this);

zero.addActionListener(this);

equal.addActionListener(this);

plus.addActionListener(this);

multiply.addActionListener(this);

divide.addActionListener(this);

minus.addActionListener(this);

dec.addActionListener(this);

clear.addActionListener(this);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==one){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("1"));

}

if(e.getSource()==two){

{

if(this.b==true){

this.b=false;

t.setText("");

}t.setText(t.getText().concat("2"));

}

}

if(e.getSource()==three){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("3"));

}

if(e.getSource()==four){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("4"));

}

if(e.getSource()==five){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("5"));

}

if(e.getSource()==six){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("6"));

}

if(e.getSource()==seven){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("7"));

}

if(e.getSource()==eight){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("8"));

}

if(e.getSource()==nine){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("9"));

}

if(e.getSource()==zero){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("0"));

}

if(e.getSource()==dec){

if(this.b==true){

this.b=false;

t.setText("");

}

t.setText(t.getText().concat("."));

}

if(e.getSource()==plus)

t.setText(t.getText().concat("+"));

if(e.getSource()==multiply)

t.setText(t.getText().concat("\*"));

if(e.getSource()==divide)

t.setText(t.getText().concat("/"));

if(e.getSource()==minus)

t.setText(t.getText().concat("-"));

if(e.getSource()==clear)

{

t.setText("");

a=false;

}

if(e.getSource()==equal)

{

this.s1=t.getText();

if("".equals(s1))

{

JOptionPane.showMessageDialog(th, "Please enter expression");

}

else {

this.a=true;

this.b=true;

}

}

}

public void view\_jframe(String ans){

t.setText(ans);

}

public void view\_cmdline(String exp, String ans){

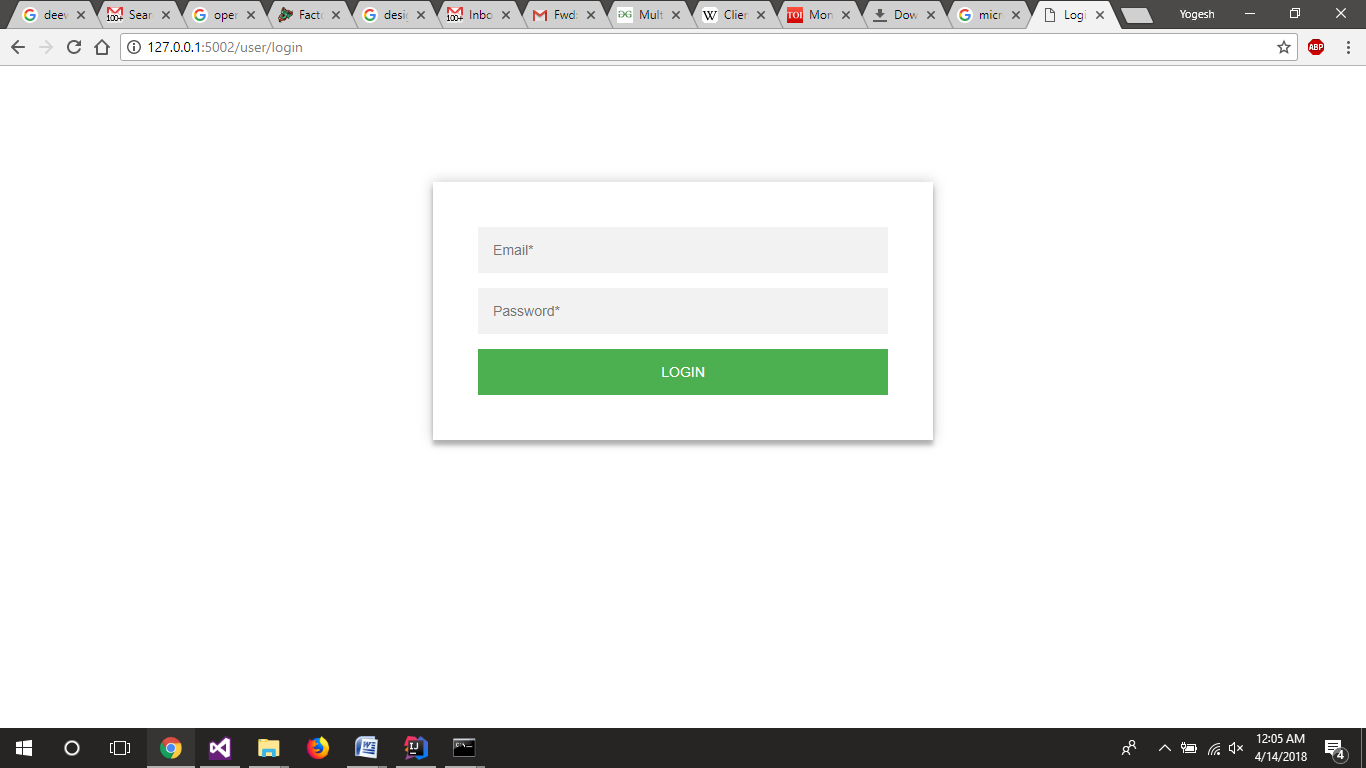
System.out.println(exp+" = "+ans);

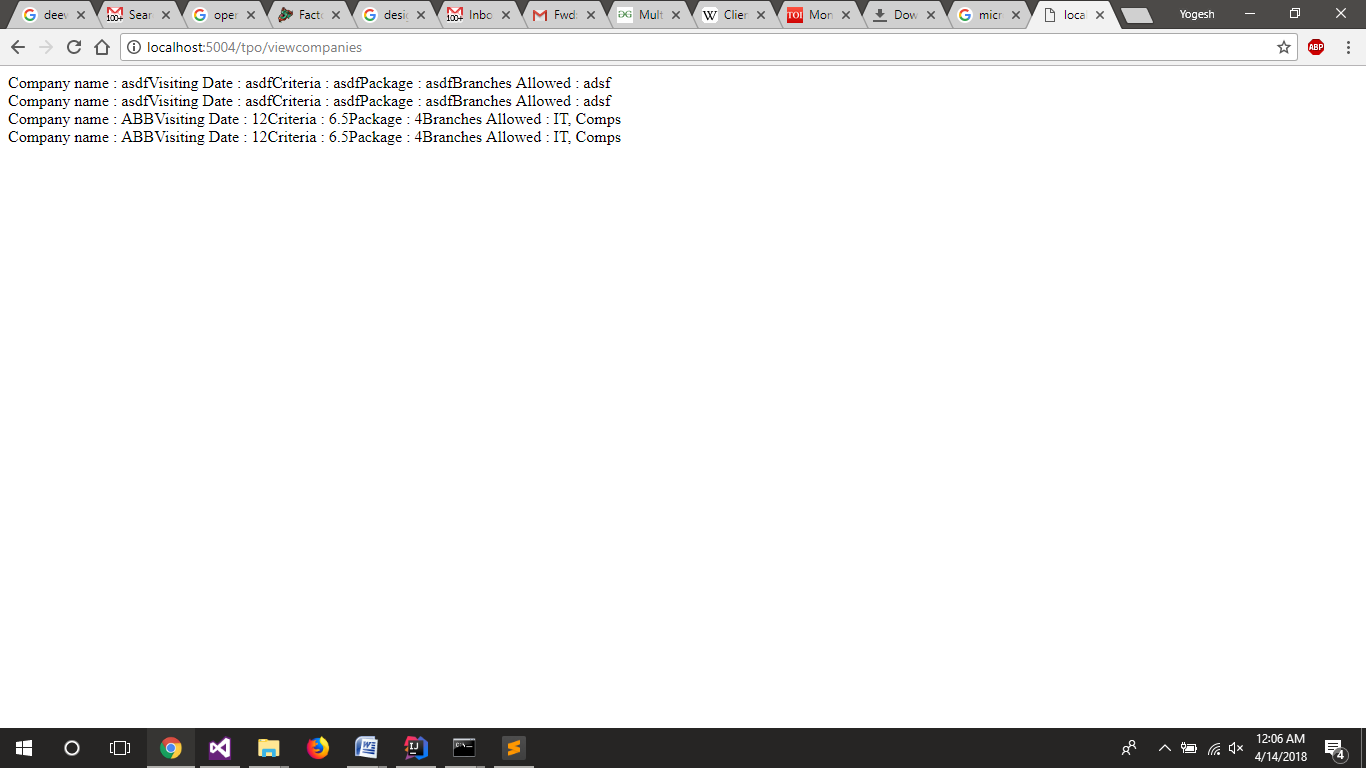
}

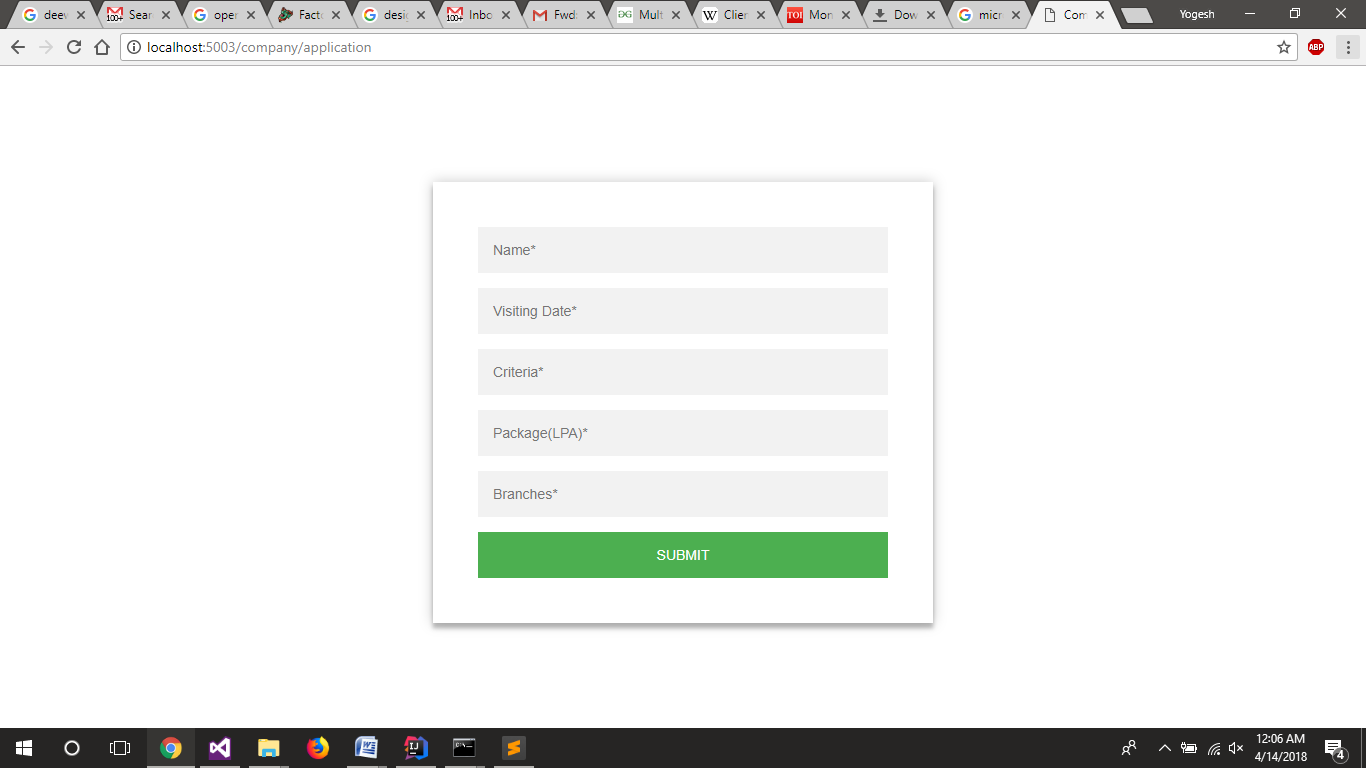
}

**Output**









**Observation:**

Thus the microservice architecture was implements using flask in python. While implementing the routing in flask had to be learnt in order to navigate the user to the correct web page and to navigate the request from users as well as other services to the correct service. Overall it was very easy once the routing was understood.

**Tutorial No. 6**

**Problem statement:**

Implement an Expression Evaluator Using Compositedesign pattern.

**Design Assumptions:**

**Composite Design Pattern**

Intent

* Compose objects into tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
* Recursive composition
* "Directories contain entries, each of which could be a directory."
* 1-to-many "has a" up the "is a" hierarchy

Problem

Application needs to manipulate a hierarchical collection of "primitive" and "composite" objects. Processing of a primitive object is handled one way, and processing of a composite object is handled differently. Having to query the "type" of each object before attempting to process it is not desirable.

Discussion

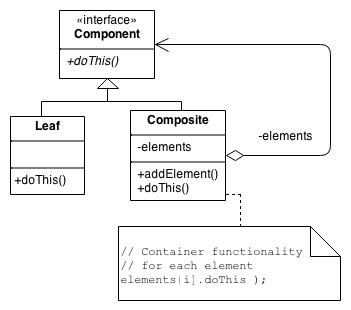
Define an abstract base class (Component) that specifies the behavior that needs to be exercised uniformly across all primitive and composite objects. Subclass the Primitive and Composite classes off of the Component class. Each Composite object "couples" itself only to the abstract type Component as it manages its "children".

Use this pattern whenever you have "composites that contain components, each of which could be a composite".

Child management methods [e.g. addChild(), removeChild()] should normally be defined in the Composite class. Unfortunately, the desire to treat Primitives and Composites uniformly requires that these methods be moved to the abstract Component class. See the "Opinions" section below for a discussion of "safety" versus "transparency" issues.

Structure

Composites that contain Components, each of which could be a Composite.



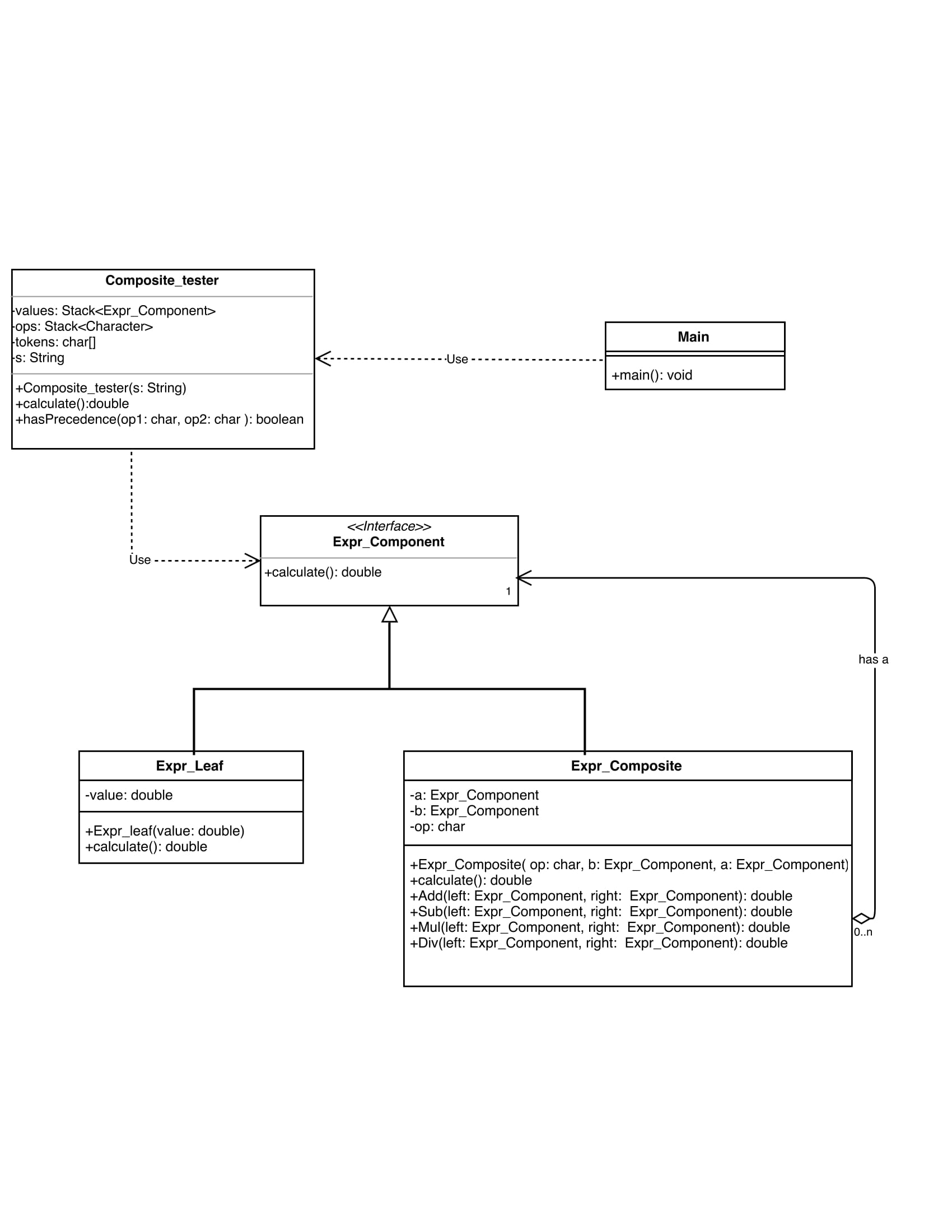
Menus that contain menu items, each of which could be a menu.

Row-column GUI layout managers that contain widgets, each of which could be a row-column GUI layout manager.

Directories that contain files, each of which could be a directory.

Containers that contain Elements, each of which could be a Container.

**Design Diagrams:**

****

**Code:**

**Composite\_tester.java**

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

package tutorial.pkg6;

import java.util.Stack;

/\*\*

\*

\* @author Jatin

\*/

public class Composite\_tester {

private Stack<Expr\_Component> values = null;

private Stack<Character> ops = null;

private char[] tokens = null;

private String s="";

public Composite\_tester(String s){

this.s=s;

}

public double calculate(){

tokens = s.toCharArray();

values = new Stack<>();

ops = new Stack<>();

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

if (tokens[i] == ' ')

continue;

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

StringBuffer sbuf = new StringBuffer();

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

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

i--;

values.push(new Expr\_leaf(Double.parseDouble(sbuf.toString())));

}

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

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

values.push(new Expr\_leaf((new Expr\_Composite(ops.pop(), values.pop(), values.pop())).calculate()));

ops.push(tokens[i]);

}

}

while (!ops.empty())

values.push(new Expr\_leaf((new Expr\_Composite(ops.pop(), values.pop(), values.pop())).calculate()));

return (values.pop()).calculate();

}

public static boolean hasPrecedence(char op1, char op2)

{

return !((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-'));

}

}

**Expr\_Component.java**

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

package tutorial.pkg6;

public interface Expr\_Component {

public double calculate();

}

**Expr\_Composite.java**

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

package tutorial.pkg6;

import java.util.Stack;

public class Expr\_Composite implements Expr\_Component{

private char op;

private Expr\_Component a;

private Expr\_Component b;

public Expr\_Composite( char op, Expr\_Component b, Expr\_Component a){

this.a=a;

this.b=b;

this.op=op;

}

@Override

public double calculate(){

switch (op)

{

case '+':

return Add(a,b);

case '-':

return Sub(a,b);

case '\*':

return Mul(a,b);

case '/':

return Div(a,b);

}

return 0;

}

public double Add(Expr\_Component left, Expr\_Component right){

return left.calculate() + right.calculate();

}

public double Sub(Expr\_Component left, Expr\_Component right){

return left.calculate() - right.calculate();

}

public double Mul(Expr\_Component left, Expr\_Component right){

return left.calculate() \* right.calculate();

}

public double Div(Expr\_Component left, Expr\_Component right){

if (right.calculate() == 0)

throw new

UnsupportedOperationException("Cannot divide by zero");

return left.calculate() / right.calculate();

}

}

**Expr\_leaf.java**

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

package tutorial.pkg6;

public class Expr\_leaf implements Expr\_Component{

double value;

public Expr\_leaf(double value) {

this.value = value;

}

@Override

public double calculate() {

return value;

}

}

**Main.java**

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

package tutorial.pkg6;

/\*\*

\*

\* @author Jatin

\*/

public class Main {

public static void main(String[] args) {

System.out.println( String.format("%.2f",(new Composite\_tester("5+6\*9")).calculate()));

}

}

**Observation:**

**Tutorial No. 7**

**Problem statement:**

Implement interpreter design pattern for Calculator program.

**Design Assumptions:**

**Interpreter Design Pattern**

Intent

* Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.
* Map a domain to a language, the language to a grammar, and the grammar to a hierarchical object-oriented design.

Problem

A class of problems occurs repeatedly in a well-defined and well-understood domain. If the domain were characterized with a "language", then problems could be easily solved with an interpretation "engine".

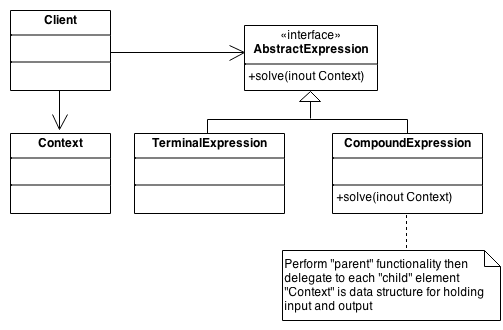
Discussion

The Interpreter pattern discusses: defining a domain language (i.e. problem characterization) as a simple language grammar, representing domain rules as language sentences, and interpreting these sentences to solve the problem. The pattern uses a class to represent each grammar rule. And since grammars are usually hierarchical in structure, an inheritance hierarchy of rule classes maps nicely.

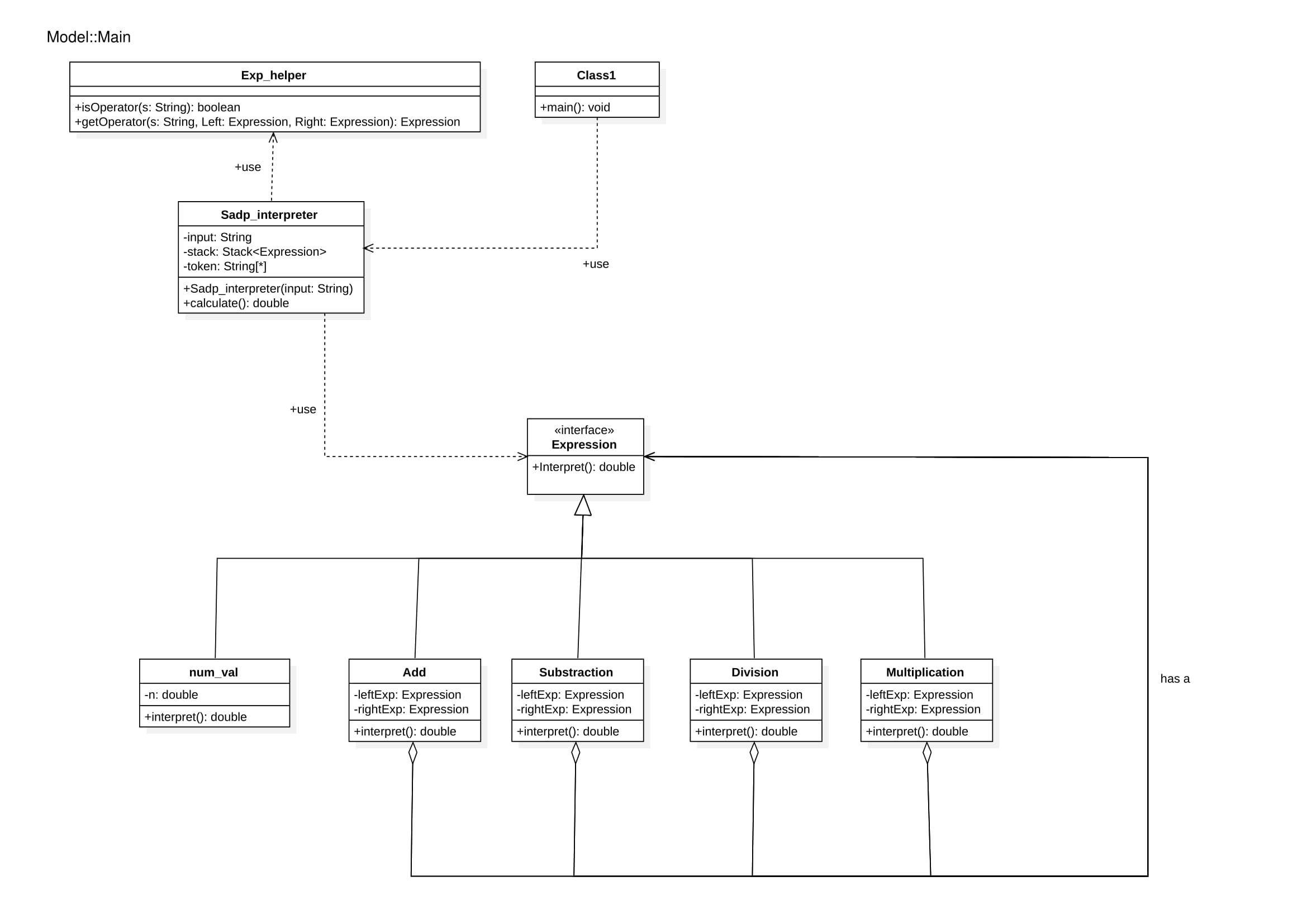
An abstract base class specifies the method interpret(). Each concrete subclass implements interpret() by accepting (as an argument) the current state of the language stream, and adding its contribution to the problem solving process.

Structure

Interpreter suggests modeling the domain with a recursive grammar. Each rule in the grammar is either a 'composite' (a rule that references other rules) or a terminal (a leaf node in a tree structure). Interpreter relies on the recursive traversal of the Composite pattern to interpret the 'sentences' it is asked to process.

****

**Design Diagrams:**

****

**Code:**

**Add.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class Add implements Expression{

private final Expression leftExp;

private final Expression rightExp;

public Add(Expression leftExpression,Expression rightExpression ){

this.leftExp = leftExpression;

this.rightExp = rightExpression;

}

@Override

public double interpret() {

return leftExp.interpret() + rightExp.interpret();

}

}

**division.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class division implements Expression{

private final Expression leftExp;

private final Expression rightExp;

public division(Expression leftExpression,Expression rightExpression ){

this.leftExp = leftExpression;

this.rightExp = rightExpression;

}

@Override

public double interpret() {

return leftExp.interpret() / rightExp.interpret();

}

}

**Exp\_helper.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class Exp\_helper {

public static boolean isOperator(String s) {

if (s.equals("+") || s.equals("-") || s.equals("\*") || s.equals("/"))

return true;

else

return false;

}

public static Expression getOperator(String s, Expression left, Expression right) {

switch (s) {

case "+":

return new Add(left, right);

case "-":

return new Substract(left, right);

case "\*":

return new Product(left, right);

case "/":

return new division(left,right);

}

return null;

}

}

**Expression.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public interface Expression {

public double interpret();

}

c:\users\yoges\desktop\semester 6\sad alternate\tutorial 7\src\sadp\_interpreter\main.java

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class main {

public static void main(String[] args) {

System.out.println(String.format("%.2f",new Sadp\_interpreter("77 3 /").calculate()));

}

}

**num\_val.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class num\_val implements Expression{

private final double n;

public num\_val(double n){

this.n = n;

}

@Override

public double interpret() {

return n;

}

}

**Product.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class Product implements Expression{

private final Expression leftExp;

private final Expression rightExp;

public Product(Expression leftExpression,Expression rightExpression ){

this.leftExp = leftExpression;

this.rightExp = rightExpression;

}

@Override

public double interpret() {

return leftExp.interpret() \* rightExp.interpret();

}

}

**Sadp\_interpreter.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

import java.util.Stack;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class Sadp\_interpreter {

/\*\*

\* @param args the command line arguments

\*/

private static String tokenString = "";

private static Stack<Expression> stack;

private static String[] tokenArray;

public Sadp\_interpreter(String tokenString){

this.tokenString = tokenString;

stack = new Stack<>();

tokenArray = tokenString.split(" ");

}

public double calculate() {

for (String s : tokenArray) {

if (Exp\_helper.isOperator(s)) {

Expression rightExpression = stack.pop();

Expression leftExpression = stack.pop();

Expression operator = Exp\_helper.getOperator(s, leftExpression,rightExpression);

double result = operator.interpret();

stack.push(new num\_val(result));

} else {

Expression i = new num\_val(Double.parseDouble(s));

stack.push(i);

}

}

return stack.pop().interpret();

}

}

**Substract.java**

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

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package sadp\_interpreter;

/\*\*

\*

\* @author ROHIT NAVNATH JATHOT

\*/

public class Substract implements Expression{

private final Expression leftExp;

private final Expression rightExp;

public Substract(Expression leftExpression,Expression rightExpression ){

this.leftExp = leftExpression;

this.rightExp = rightExpression;

}

@Override

public double interpret() {

return leftExp.interpret() - rightExp.interpret();

}

}

**Output**

**Observation:**

Thus the interpreter design pattern was used in calculator program. We parsed the input string using interpreter pattern and produced the output.