**Tutorial No. 3**

**Problem statement:**

**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.

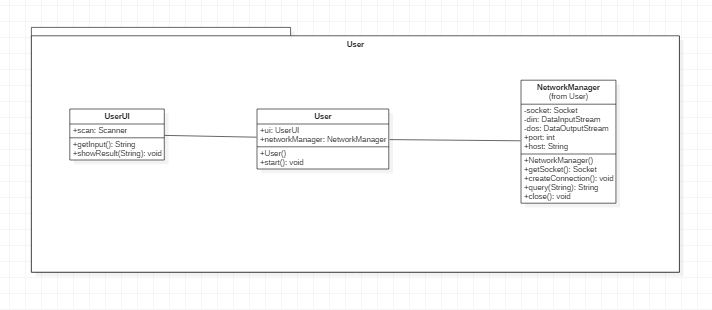
**Advantages**

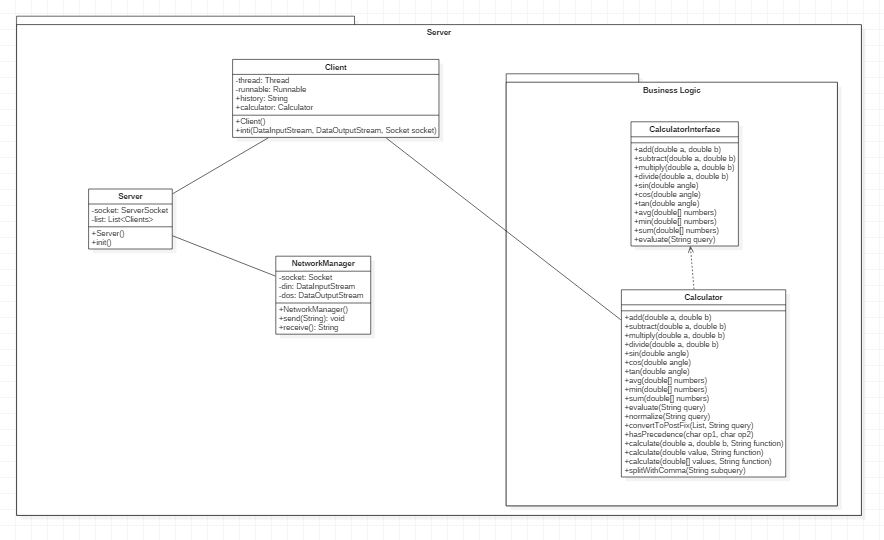
* Simultaneous development – Multiple developers can work simultaneously on the model, controller and views.
* [High cohesion](https://en.wikipedia.org/wiki/Cohesion_(computer_science)) – MVC enables logical grouping of related actions on a controller together. The views for a specific model are also 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:**

**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);

//Inverse Trigonometric

public double asin(double angle);

public double acos(double angle);

public double atan(double angle);

//Logarithmic

public double log(double number);

}

c:\users\yoges\desktop\semester 6\sad alternate\calculator\src\com\mangnaik\yogesh\calculator\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;

/\*\*

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

\*/

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);

}

**Test Cases :**

**Observation:**