**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:**

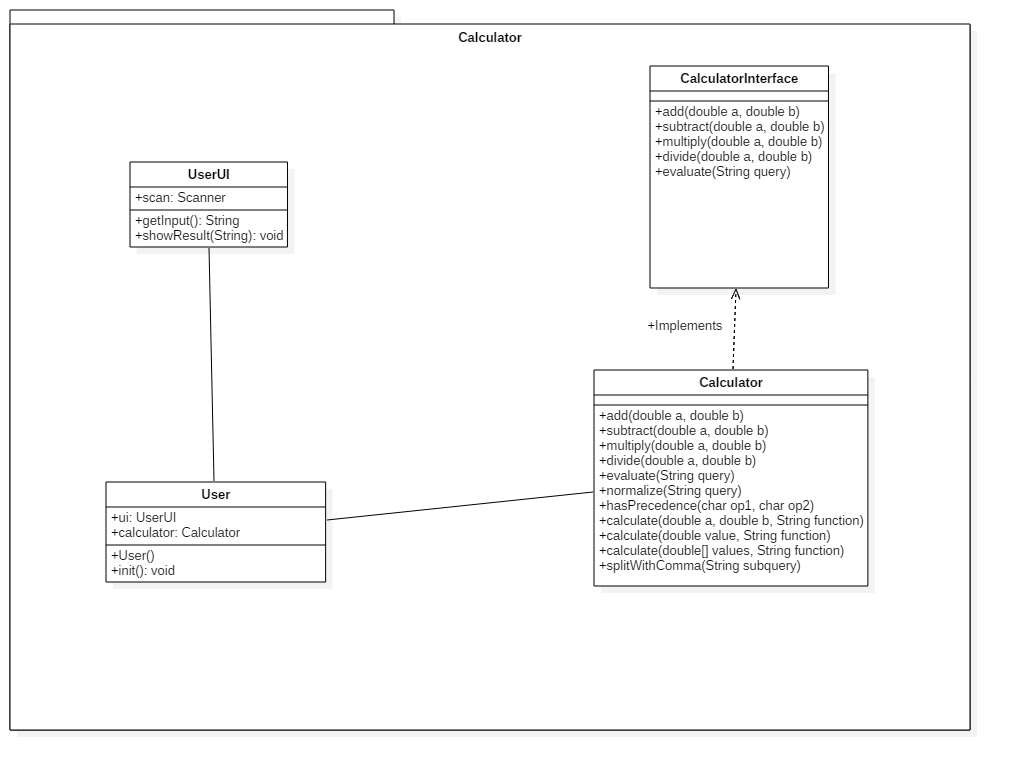
When a [bank](https://en.wikipedia.org/wiki/Bank) customer accesses [online banking](https://en.wikipedia.org/wiki/Online_banking) services with a [web browser](https://en.wikipedia.org/wiki/Web_browser) (the client), the client initiates a request to the bank's web server. The customer's [login](https://en.wikipedia.org/wiki/Login) 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.

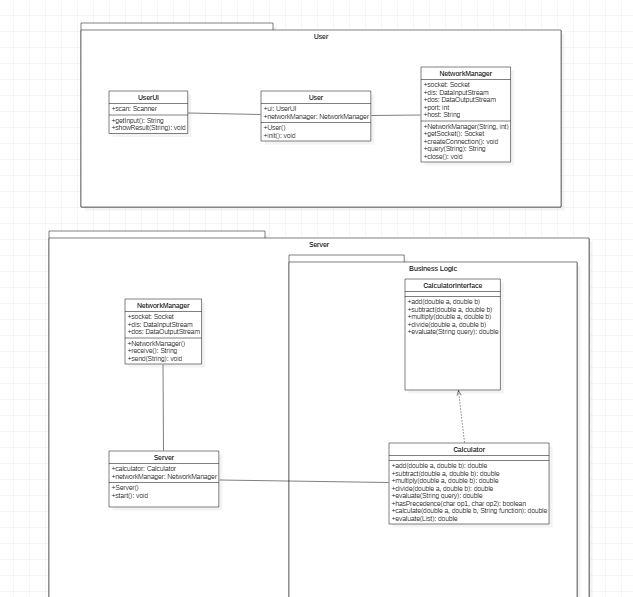
This example illustrates a [design pattern](https://en.wikipedia.org/wiki/Design_pattern) applicable to the client–server model: [separation of concerns](https://en.wikipedia.org/wiki/Separation_of_concerns).

**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();

}

}

**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.