Group members:

161081971 – Apeksha Barve

161081977 – Sayali Patil

161081978 – Pragati Patil

161081982 – Bhagyahri Kamble

**Tutorial No. 01**

**Aim:** Implement Calculator with addition, multiplication, division and subtraction operations using **Client Server**and**Monolithic** Architectures; also draw class diagrams for the architectures.

**Theory:**

**Monolithic architecture:**

1. A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means composed all in one piece.
2. Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent rather than loosely coupled as is the case with modular software programs.
3. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled.
4. there are benefits to monolithic architectures as well. Monolithic programs typically have better throughput than modular approaches, such as the micro service architecture (MSA) and they can be easier to test and debug because, with fewer elements there are fewer variables that come into play.

**Advantages of monolithic architecture:**

1. Faster Initial Development

a. With one application, it would be relatively easy to add additional features, especially when the application is relatively small.  Several features were added to the 80,000 Hours codebase relatively easily (it would have taken more code to have separate applications for each one).

2. Little User Confusion (e.g. new apps on the same platform would be more easily found by users to the existing apps)

Users wouldn't have to learn about different applications, but would be focussed towards one application.

3. Improved Integration

Features could integrate with each other well and easily, as there is only one user table.

4. User Interface Similarity

All of the pieces of the application would look very similar, so it would be obvious it's all part of one system.

**Disadvantages**

1. Substantially Less Iteration

The larger a website is, the more difficult it is to change it.  It would be incredibly tough, for example, to change the theme or UI or a monolith application.  The means that we would have significantly less experimentation. This is one reason why the 80,000 Hours social network was put on hold after the redesign; it would have vastly increased the time to actually make that happen.

1. Maintenance

The larger a website is, the more difficult it becomes to maintain the entire thing.  Maintenance costs may go up exponentially with site size.  This is one reason why many startups, with large amounts of funding (1-20 million dollars) have relatively simple websites.  Large ones, especially large ones with large feature sets, typically don't have good reputations for stability.

1. Power Centralization

Obviously if there's one monolith application, it would ultimately be owned by one person or organization.  If all application uses would be put into this application, then this is a lot of power to trust in one organization.  There's currently a decent level of distrust and conflicting goals between EA organizations.  Thus, to have one EA organization in charge of all EA apps (one large one) would present a situation that could create controversy.

**Client –Server Architecture:**

1. Client-server (C/S) or "two-tier" architecture (in comparison with a peer-to-peer architecture) has two separate types of nodes on the network: servers, that store information and clients, that send requests for information to servers.
2. Usually, but not always, a client computer and a server computer are two separate devices. A server computer contains large amounts of memory and disk space, while client computers features graphic user interface to support the display of data stored on server. There are many different types of *client/server software*however their basic architecture remains the same.

**Client/Server Benefits**

Client/server approach to networking has proven to be a cost-effective way to share data between tens or hundreds of computers. Considering the client/server computing in terms of a manager-employee relationship, the following list outlines some of the benefits and drawbacks of client/server solutions.

1. **Centralized Information Storage.**

The server stores the data and coordinates the access to information and its modification. This helps to keep the data consistent and up-to-date, even when multiple users/clients are working with it simultaneously.

1. **Delegation**

Managers or team leaders have more information, experience and knowledge about the company and day-to-day operations. Their deep understanding of the business processes, priorities, strategy, goals, and important tasks allows them to easily share information as needed and delegate work to their employees.

1. **Focus**

The employees or team members may have less knowledge and experience or their vision of the strategy and goals is far from clear. By using [client/server software](http://en.wikipedia.org/wiki/Client%E2%80%93server_model) they have more focused tasks and also they might get a clear picture of company strategy if their team leaders provide them with access to such information.

1. **Collaboration**

This is the essential aspect of how client/server computing works. It lets the manager plan, assign and notify the team members of their tasks. Once finished with their work, employees report the results back to the manager for verification.

1. **Security**

Very often the database is securely locked away from unauthorized access and its data is denied 'view' and/or 'edit' permission, that prevents violations from outside and inside the office.

**Requirement Specification:**

This section describes the external influences imposed on the calculator program.

**Hardware and Software:**

The program shall be written in standard JAVA, as compiled by the JAVA JDK. The program shall use only standard JAVA library functions. The program shall be usable on any system which supports the compiler, and shall not require any particular hardware or software. 2.2

**External Data Bases and File Interfaces:**

There are no existing external data bases or files that will be needed by this program.

**Human Interfaces:**

The program shall operate as much as possible in the same way as a regular handheld calculator, such that anyone familiar with the operation of such a device should have little trouble using the program. Once the program is started, the program shall produce a prompt. The user can then type a sequence of numbers and operators, similar to a regular calculator. The program will display the calculated answer to the entered problem on the line following the input line. Some examples that show the proper function of the calculator follow below (the computer-generated output is underlined):

1.47\*3.5+15.92=26.9024

**Functions:**

The following section outlines all of the functions required of the calculator program. 3.1 Data Flow Diagram Figure 1 is a data flow diagram which identifies the major functions required of the calculator program and shows their relationship to each other. The numbers included with each function are keyed to the subsections

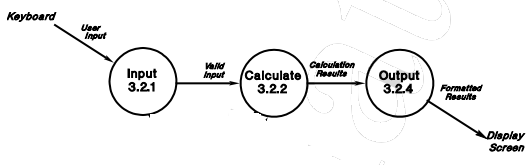
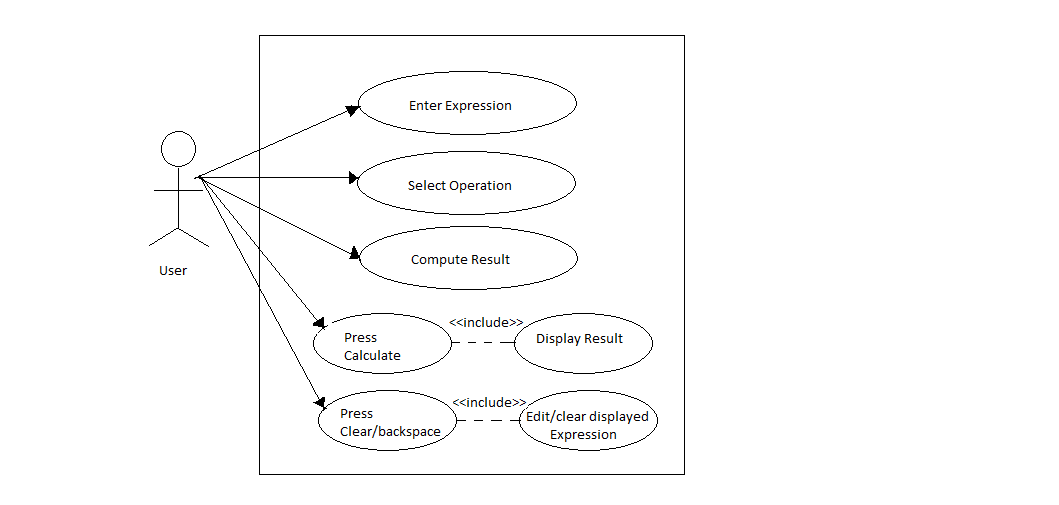
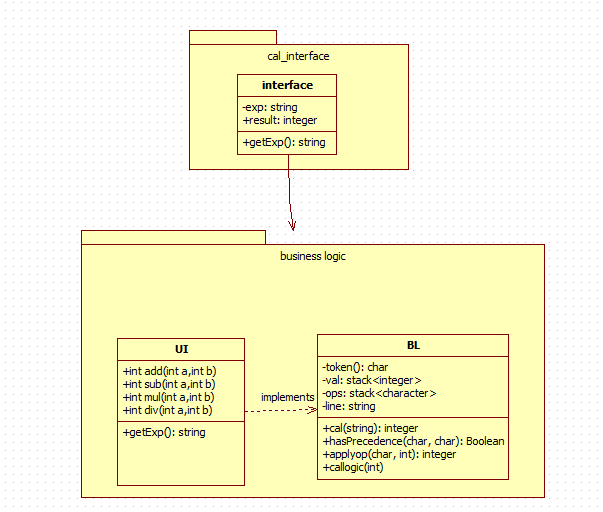


Fig.a. Data flow diagram for monolithic architecture

**Use case diagram:**



Class Diagram:



**Calculator code:**

**client\_side(user interface)**

**package userInterface;**

**import java.net.\*;**

**import java.io.\*;**

**public class Client**

**{**

**private Socket socket = null;**

**private DataInputStream input = null;**

**private DataOutputStream out = null;**

**private DataInputStream in = null;**

**public Client(String address, int port)**

**{**

**try**

**{**

**socket = new Socket(address, port);**

**System.out.println("Connected");**

**input = new DataInputStream(System.in);**

**out = new DataOutputStream(socket.getOutputStream());**

**in = new DataInputStream(socket.getInputStream());**

**}**

**catch(UnknownHostException u)**

**{**

**System.out.println(u);**

**}**

**catch(IOException i)**

**{**

**System.out.println(i);**

**}**

**String line = "";**

**String ch;**

**while(true)**

**{**

**try**

**{**

**System.out.println("Enter Expression...");**

**line = input.readLine();**

**out.writeUTF(line);**

**line = in.readUTF();**

**System.out.println(line + "\nContinue (Y/N)?:");**

**ch = input.readLine();**

**if(ch.equals("N") || ch.equals("n")){**

**out.writeUTF("Over");**

**break;**

**}**

**}**

**catch(IOException i)**

**{**

**System.out.println(i);**

**}**

**}**

**try**

**{**

**input.close();**

**out.close();**

**socket.close();**

**}**

**catch(IOException i)**

**{**

**System.out.println(i);**

**}**

**}**

**public static void main(String args[])**

**{**

**Client client = new Client("127.0.0.1", 5000);**

**}**

**}**

sserver\_side(business logic):

package businessLogic;

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Server

{

private Socket socket = null;

private ServerSocket server = null;

private DataInputStream in = null;

private DataOutputStream out = null;

private BusinessLogic.calLogic calc;

private String line = "";

private char[] tokens;

private Stack<Integer> val = new Stack<Integer>();

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

public Server(int port)

{

try

{

server = new ServerSocket(port);

System.out.println("Server started");

System.out.println("Waiting for a client ...");

socket = server.accept();

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

in = new DataInputStream(

new BufferedInputStream(socket.getInputStream()));

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

calc = new BusinessLogic.calLogic();

while (true)

{

try

{

line = in.readUTF();

System.out.println(line);

if(line.equals("Over") || line.equals("over"))

break;

int result = calc.cal(line);

System.out.println(result);

out.writeUTF(Integer.toString(result));

}

catch(IOException i)

{

System.out.println(i);

}

}

System.out.println("Closing connection");

socket.close();

in.close();

}

catch(IOException i)

{

System.out.println(i);

}

}

public static void main(String args[])

{

Server server = new Server(5000);

}

}