**Tutorial No. 1**

**Aim** :- Implement a calculator based on monolithic and client-server architecture.

**Theory** :-

**Monolithic Architecture**

In software engineering, a monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform. Monolithic, in this context, means composed all in one piece. 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. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled.

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; 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.

**Client-Server Architecture**

In Computer science, client-server is a software architecture model consisting of two parts, client systems and server systems, both communicating over a computer network or on the same computer. A client-server application is a distributed system made up of both client and server software. Client server

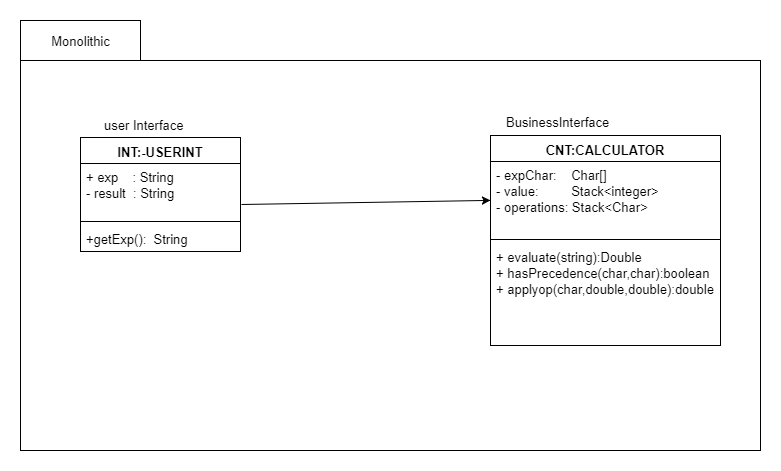
application provide a better way to share the workload. The client process always initiates a connection to the server, while the server process always waits for requests from any client. When both the client process and server process are running on the same computer, this is called a single seat setup.

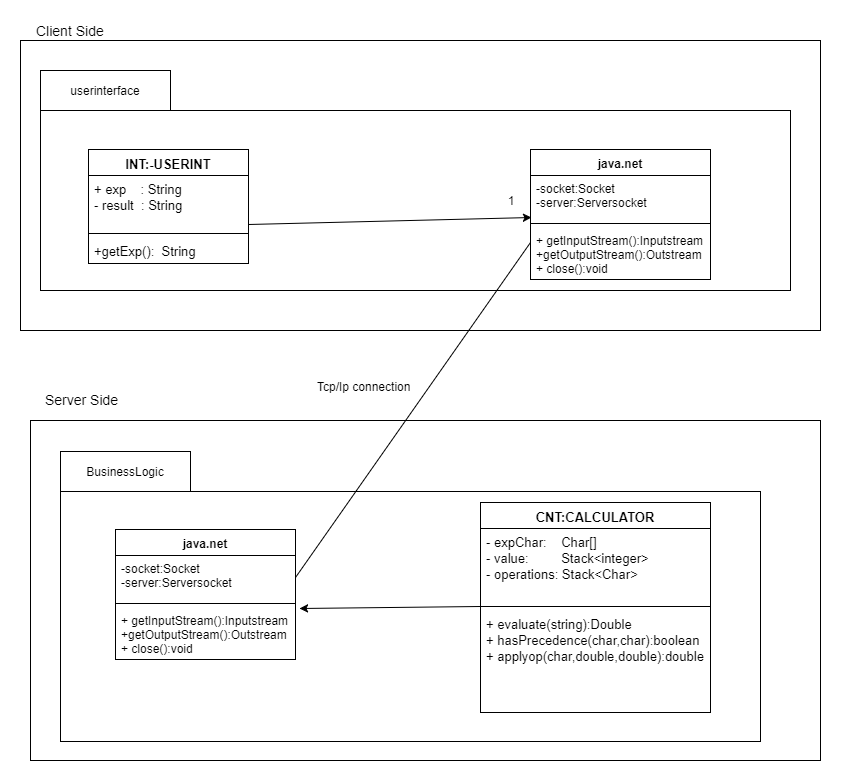
Another type of related software architecture is known as peer-to-peer, because each host or application instance can simultaneously act as both a client and a server (unlike centralized servers of the client-server model) and because each has equivalent responsibilities and status. Peer-to-peer architectures are often abbreviated using the acronym P2P.

The client-server relationship describes the relation between the client and how it makes a service request to the server, and how the server can accept these requests, process them, and return the requested information to the client. The interaction between client and server is often described using sequence diagrams. Sequence diagrams are standardized in the Unified Modeling Language.

The basic type of client-server software architecture employs only two types of hosts: clients and servers. This type of architecture is sometimes referred to as two-tier. The two-tier architecture means that the client acts as one tier and server process acts as the other tier. The client-server software architecture has become one of the basic models of network computing. Many types of applications have been written using the client-server model. Standard networked functions such as E-mail exchange, web access and database access, are based on the client-server model. For example, a web browser is a client program at the user computer that may access information at any web server in the world.

**Class Diagram** :-





Test plan:

The following is a list of the items to be tested:

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

Some more modules were included and changed during development phase, which are not fully recognized or documented yet. So, test items of those modules will be included in the next version of the test plan.

**Features to be tested**

1. **Expression** 
   1. Test Id.: cal.0
   2. Description: To check correct functioning of the Calculate Module
   3. Pre-Condition: Calculator Ui must be loaded and Input must be provided
   4. Equivalence Classes Evaluate field:
      1. Length of expression :
         1. Length must be limited till 2^20
      2. Use of other function
         1. Expression containing function other than algebraic not allowed
         2. Expression containing characters as a input are not allowed
      3. Undefined in math
         1. Expression containing divide by 0 are not allowed as it is not defined in math
      4. Closing of brackets
         1. Expression must be with all closing of the brackets
      5. Use of two function without number in between
2. Expression must be with successive operation and numbers in between i.e use of two functions successively is not allowed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No** | **Input** | **Steps** | **Expected Output** | **Actual Output** | **Result** |
| 1 | Case 1  :- 4+9+85 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 94 | 94 | pass |
| 2 | Case 1  :- 20-10 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 10 | 10 | pass |
| 3 | Case 1  :- 5\*6 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 30 | 30 | Pass |
| 4 | Case 1  :- 6/3 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 2 | 2 | Pass |
| 5 | Case 2  :-9\*6-3 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 51 | 51 | Pass |
| 6 | Case 2  :- 9/3+2 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 5 | 5 | Pass |
| 7 | Case 2  :- 9\*4/12 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 3 | 3 | Pass |
| 8 | Case 3  :- (2+3)\*5 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 25 | 25 | Pass |
| 9 | Case 3:-  (6-4)/2 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 1 | 1 | Pass |
| 10 | Case 4:-  4\*5+6/3\*5 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 30 | 30 | Pass |
| 11 | Case:-2  (20\*3)/4-52\*(10+6/3) | 1)Compile CalUi.java file  2)Enter expression  3) run file | -609.0 | -609.0 | Pass |
| 12 | Case:-2  5.5/1.2\*3.1 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 14.2083 | 14.2083 | Pass |
| 13 | Case:-1  10/3 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 3.333 | 3.333 | Pass |
| 14 | Case:-  10\*2.3+9-1 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 31.0 | 31.0 | Pass |
| 15 | Case:-  (5.6/8)\*5-0.2 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 3.3 | 3.3 | Pass |
| 16 | Case:-  (5.5/1.2)\*3.1 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 14.2083 | 14.2083 | Pass |
| 17 | Case:-  15\*0.0 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 0.00 | 0.0 | Pass |
| 18 | Case:-  15/0 | 1)Compile CalUi.java file  2)Enter expression  3) run file | Infinite | Exception | Fail |
| 19 | Case:-  5.5/1.2\*3.1 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 14.2083 | 14.2083 | Pass |
| 20 | Case:-  (4+6)/5-(5\*1 | 1)Compile CalUi.java file  2)Enter expression  3) run file | -3.0 | Error Display for  Not closing bracket | Fail |
| 21 | Case:-  8\*2+/3 | 1)Compile CalUi.java file  2)Enter expression  3) run file | 19 or  5.6 | Error Display for  Using two brackets at successive | Fail |
| 22 | Case:-  80/77-(73\*96+89)\*3-56 | 1)Compile CalUi.java file  2)Enter expression  3) run file | -21345.9610 | -21345.9610Display for  Not closing bracket | Pass |

**Conclusion**:- Thus we studied Monolithic Architecture and client server architecture and implemented same for calculator also tested using junit4 .