Rob Wilkie B00684037

Assignment 2: MathApp

COM577

Introduction

This assignment is concerned with the development of several types of client-server system. For both the client and server, generic start-points are used for convenience and ease of testing, the server starting point asks the user which type of server to start: iterative, concurrent or HTTP. Likewise, the client starting point asks the user do they want to start a socket client or an HTTP client. The overall architecture is shown in the following diagram (figure 1).

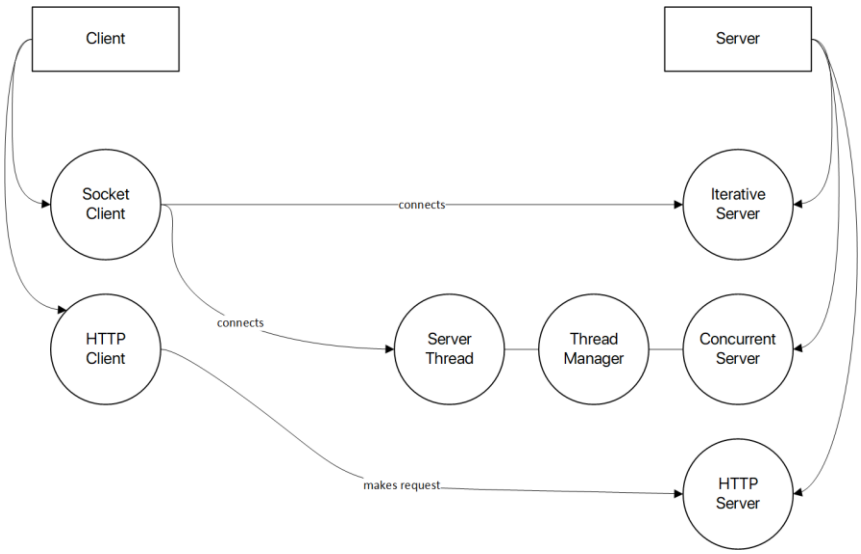


Figure 1: Overall Architecture

Technically, the socket client connects to the concurrent server via a ServerSocket, but in practice the ServerThread handles the socket connection to the client. This is depicted in detail later in figure 3.

This architecture enables all the common code to be shared across the different clients and servers. The project structure across all client and server types implemented in this assignment is given below.

```
mathapp
    Client.java
    Server.java

    common
        ClientBase.java
        ServerBase.java
        Colors.java
        Constants.java
        Logger.java
        MathService.java
        Params.java
        Response.java
        ResponseType.java

    http
        client
            HttpClient.java

        server
            HTTPServer.java

    socket
        IOSocket.java

        client
            SocketClient.java

        server
            Request.java
            ServerConnection.java
            ServerConnectionLog.java

            concurrent
                ConcurrentServer.java
                ServerThread.java
                ThreadManager.java

            iterative
                IterativeServer.java
```

On the client side, the assignment requirement to *"close the communication with the server after receiving one result"* has been interpreted as a user option so that should the user wish to perform several calculations before closing the client; then they can. If only one calculation is needed then the user can choose to close the client at that point. The way in which the user expresses the calculation they wish to have performed has not been specified in the requirements of the assignment and so an intuitive scheme has been adopted. The protocol for conveying the calculation between client and server follows the precise specification given in the assignment requirements e.g. `+:6.7:3.2`.

With reference to the tree structure above, appendix A contains a full listing of all classes in the mathapp root and common packages. Appendix B contains listings for the socket solution with appendix B.1 containing listings for the iterative server and B.2 containing listings for the concurrent server. Appendix B.3 contains the socket-based client which is shared between both iterative and concurrent servers. Appendix C contains source code for the HTTP server and client.

Question 1: Socket-based Solution

In the following sections reference is made to "IOSocket"; this is a class that wraps a Socket and provides functionality to send and receive strings of text.

Client

When the client starts, it instantiates a new IOSocket using a port number in common with the server it wishes to connect to. When the server accepts the connection, initially a confirmation of connection is received from the server. The client manages dialog with the user to obtain the maths calculation details, it builds a Params object which contains the maths calculation required and sends a stringified version of the calculation to the server via the socket.

A significant amount of validation of the user input takes place in the client, this is achieved through the `getValidInput()` and `getYesNo()` methods inherited from ClientBase.

The client waits for the calculation result to be returned and then displays this on the console for the user. At this point the user is prompted if they would like to perform another calculation and if not, the connection is closed and the client closes.

This client is used to connect to both the iterative and concurrent servers. The socket client's source code is given in appendix B.3. The client-side message sequence for connecting to a server is depicted in both figure 2 (iterative server) and figure 3 (concurrent server) since the same client logic is used in both of these servers.

```
[17:51:58.890647000] [SYSTEM]    Which type of client do you wish to run?
[17:51:58.904609900] [SYSTEM]    [1] Socket
[17:51:58.904609900] [SYSTEM]    [2] HTTP
                >>> 1
[17:52:00.904487300] [CLIENT]    Attempting to connect to server on port 4628
[17:52:00.914483500] [SERVER]    Connected
[17:52:00.915458200] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                >>> 54*64
[17:52:07.087195600] [SERVER]    Result: 3456.0
[17:52:07.088193500] [CLIENT]    Do you want to do another calculation? y/n
                >>> n
[17:57:03.678209400] [CLIENT]    Connection closed
[17:57:03.678209400] [CLIENT]    Client closing
```

Iterative Server

The iterative server (appendix B.1) handles both connection requests and transactions from a client in a simple manner which is depicted in the UML sequence diagram below (figure 2).

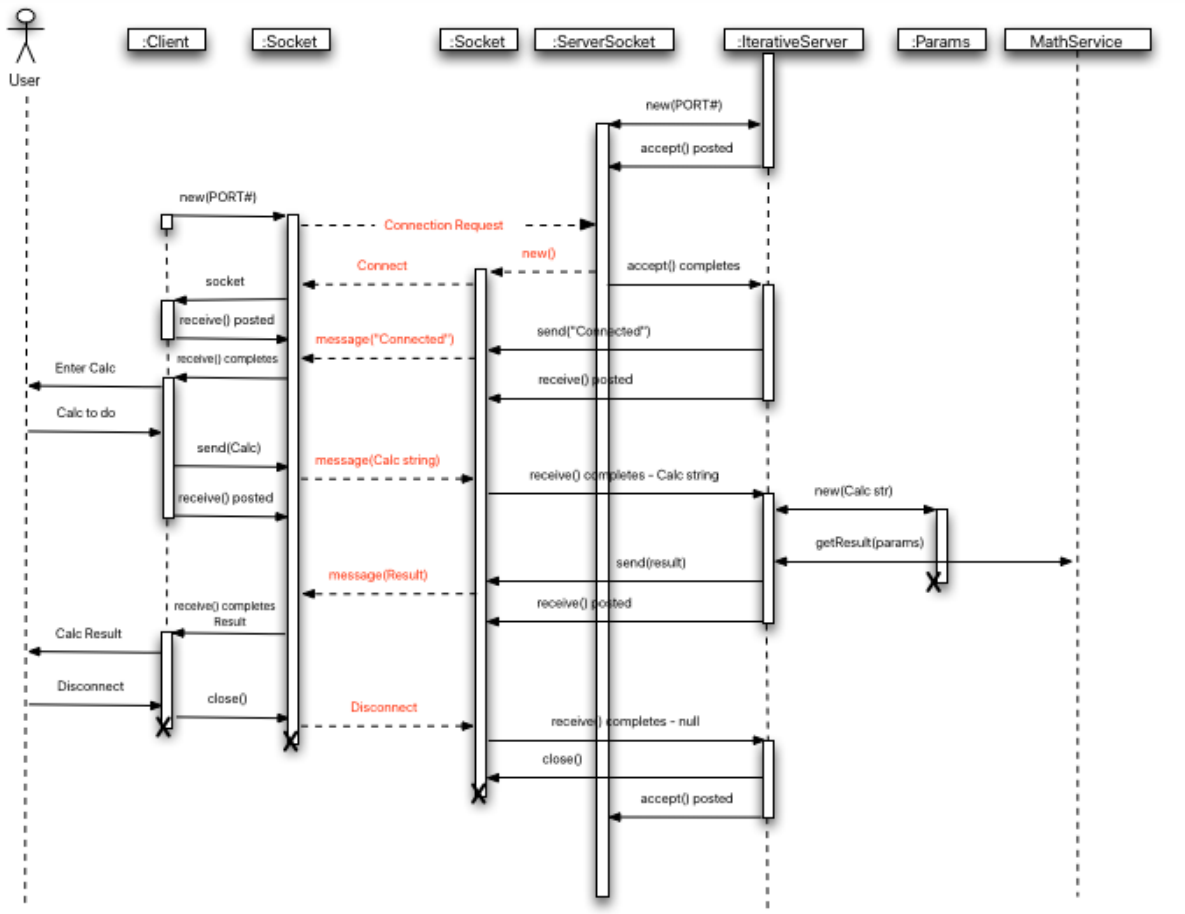


Figure 2: Iterative Server UML Sequence Diagram

When the server starts, it establishes a ServerSocket bound to a port and listens for an incoming client connection. When a client connects; the server sends an initial connection confirmation message to the client and then the client can make maths requests to server. The server makes use of the MathService class to perform the necessary calculations and return the results. During this period no further connections are accepted. When the client disconnects then the server goes back to listening for another incoming client connection. If two or more clients attempt to connect, the ServerSocket handles the queue of requests (up to 50 by default). When the current client disconnects, the next client connection request on the socket is accepted. This makes the iterative server a bottleneck if there are many clients requesting to connect and therefore limits its value.

```
[17:42:47.447209500] [SYSTEM]    Which type of server do you wish to run?
[17:42:47.459184900] [SYSTEM]    [1] Iterative
[17:42:47.459184900] [SYSTEM]    [2] Concurrent
[17:42:47.459184900] [SYSTEM]    [3] HTTP
                >>> 1

[17:42:52.828988000] [SERVER]    Iterative server listening on port 4628
[17:43:00.706002900] [SERVER]    [C1] Client connected from 127.0.0.1:63963
[17:43:22.059677200] [SERVER]    [C1R1] -:135.0:35.0 (135.0 - 35.0) Result: 100.0
[17:51:36.160232800] [SERVER]    [C1R2] ^:654.0:3.0 (654.0 ^ 3.0) Result: 2.79726264E8
[17:51:57.894114900] [SERVER]    [C1] Client disconnected
[17:52:00.913463900] [SERVER]    [C2] Client connected from 127.0.0.1:65470
[17:52:07.087195600] [SERVER]    [C2R1] *:54.0:64.0 (54.0 * 64.0) Result: 3456.0
```

The screenshot on the left depicts two clients sequentially connecting to the same iterative server, the first client can be seen making two calculation requests, with the second client only making one.

Concurrent Server

The concurrent server gets around the limitations of the iterative server by delegating responsibility for servicing a client's needs to a dedicated child server thread (ServerThread class). So, when the concurrent server starts up, it dedicates itself to listening for client connection requests on the ServerSocket. When a new client connection is received by the server, the server spins up a new ServerThread instance and passes the socket reference for the current client connection to it. From then on, the ServerThread instance handles all of that client's requests. The concurrent server then immediately goes back to listening for new connections.

The UML sequence diagram below (figure 3) depicts the interaction sequence for a client connecting to the concurrent server.

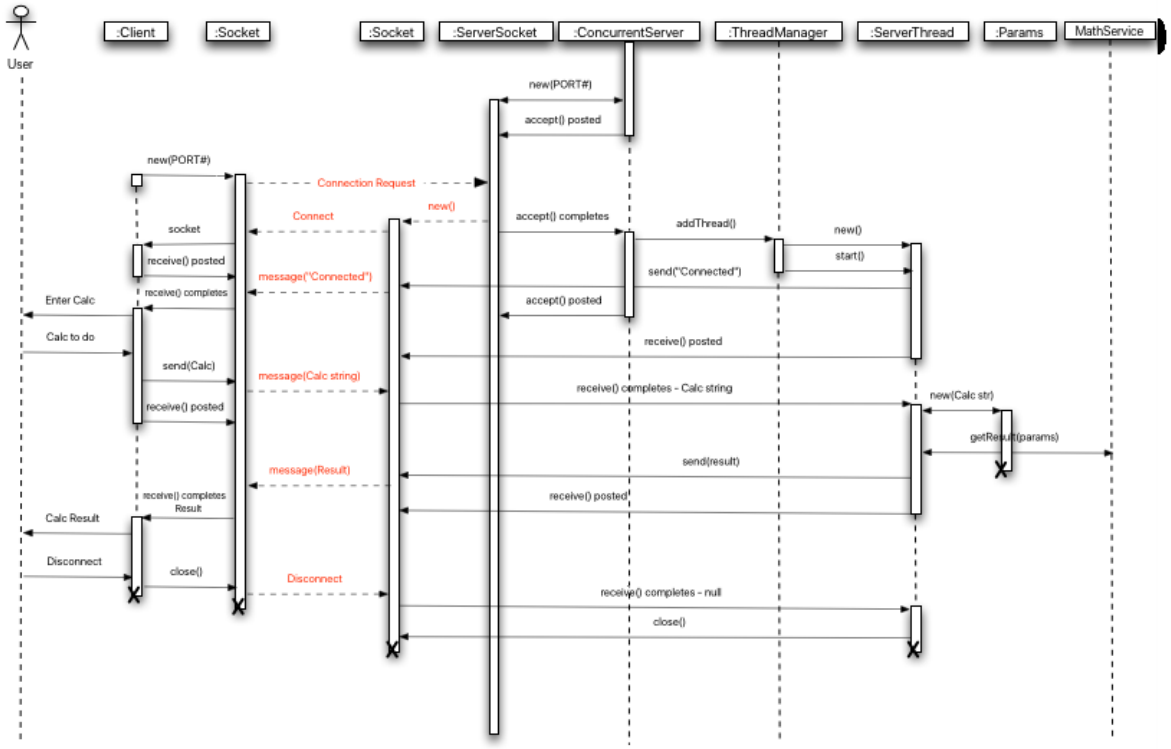


Figure 3: Concurrent Server UML Sequence Diagram

The ServerThread is responsible for all communication with the client. It sends a connection confirmation, receives the data from client, calls the MathService and returns the calculation result back to the client. When a client disconnects, the thread interrupts itself, causing it to terminate.

```
[18:20:44.983007300] [SERVER]    Concurrent server listening on port 4628
[18:20:55.922048700] [SERVER]    [C1] Client connected from 127.0.0.1:53237
[18:20:55.923046100] [SERVER]    [C1] Starting worker thread
[18:20:55.923046100] [WORKER]    [C1] Worker thread started
[18:21:00.809583100] [WORKER]    [C1R1] -:545.0:21.0 (545.0 - 21.0) Result: 524.0
[18:21:16.095686800] [SERVER]    [C2] Client connected from 127.0.0.1:53293
[18:21:16.095686800] [SERVER]    [C2] Starting worker thread
[18:21:16.095686800] [WORKER]    [C2] Worker thread started
[18:21:41.062431200] [WORKER]    [C1R2] ^:2.0:5.0 (2.0 ^ 5.0) Result: 32.0
[18:21:48.397155600] [WORKER]    [C2R1] /:54.0:6.0 (54.0 / 6.0) Result: 9.0
[18:21:54.506147900] [SERVER]    [C2] Client disconnected
```

The screenshot on the left depicts two clients connecting to the server. Separate worker threads are created to service each individual client and the screen shows the two clients making concurrent requests.

Testing and Evaluation

This section provides a set of test cases used to verify the iterative and concurrent servers. The iterative and concurrent servers share common code for the maths calculations and the validation of user input, and so only one set of tests is performed on each of these categories. The tests have therefore been split into four categories:

1. iterative server tests
2. concurrent server tests
3. maths calculation tests
4. user input validation tests

Test results where appropriate are provided in a following section.

No.	Description	Expected Result	Actual Result	Pass/Fail
Iterative Server Tests				
A1	Client A connects to server	Server log indicates client has connected; client log also indicates it is connected	As expected	P
A2	Client A disconnects from server	Server log indicates client has disconnected	As expected	P
A3	While client A is connected to server, client B attempts to connect	Client B should suspend waiting for client A to disconnect, when client A disconnects, client B should immediately connect to the server	As expected	P
Concurrent Server Tests				
B1	Client A connects to server	Server log indicates client has connected; client log also indicates it is connected	As expected	P
B2	Client A disconnects from server	Server log indicates client has disconnected	As expected	P
B3	While client A is connected to server, client B attempts to connect	Client B should immediately connect so that both clients A and B are simultaneously connected to the server	As expected	P
B4	While clients A and B are connected to the server, client B can disconnect and client A can continue to make requests	Client A can continue to make requests after client B disconnects	As expected	P
Maths Calculation Tests				
C1	Valid calculation using + symbol 127 + 16	Result = 143	As expected	P
C2	Valid calculation using – symbol 743 – 287	Result = 456	As expected	P
C3	Valid calculation using * symbol 41 * 59	Result = 2419	As expected	P
C4	Valid calculation using / symbol 540 / 90	Result = 6	As expected	P

C5	Valid calculation using ^ symbol 2 ^ 8	Result = 256	As expected	P
User Input Validation Tests				
D1	Selection of "yes" to perform another calculation	User is prompted to enter another calculation	As expected	P
D2	Selection of "no" to not perform another calculation	Client should close	As expected	P
D3	User enters "t" when prompted for y/n	Any response other than y/n will result in the user being re-prompted	As expected	P
D4	Invalid calculation – user inputs "a * 4"	User should be warned no alphabetical characters are permitted	As expected	P
D5	Missing argument – user inputs "500 *"	User should be warned they did not enter a valid calculation	As expected	P
D6	Missing operator – user inputs "123 123"	User should be warned they have not entered an operator	As expected	P
D7	Invalid operator – user inputs "12 % 2"	User should be warned they have entered an invalid operator	As expected	P
D8	Duplicate operator – user inputs "123 ++ 456"	User should be warned they entered more than one operator	As expected	P
D9	Invalid calculation – user inputs "+ 123 123"	User should be warned there is something wrong	As expected	P

Test Results

Test A1

```
[19:12:40.888472200] [SYSTEM]    Which type of server do you wish to run?
[19:12:40.902434700] [SYSTEM]    [1] Iterative
[19:12:40.902434700] [SYSTEM]    [2] Concurrent
[19:12:40.902434700] [SYSTEM]    [3] HTTP
                                                >>> 1

[19:12:43.480643500] [SERVER]    Iterative server listening on port 4628
[19:12:48.386430400] [SERVER]    [C1] Client connected from 127.0.0.1:50884
```

Test A2

```
[19:12:40.888472200] [SYSTEM]    Which type of server do you wish to run?
[19:12:40.902434700] [SYSTEM]    [1] Iterative
[19:12:40.902434700] [SYSTEM]    [2] Concurrent
[19:12:40.902434700] [SYSTEM]    [3] HTTP
                                                >>> 1

[19:12:43.480643500] [SERVER]    Iterative server listening on port 4628
[19:12:48.386430400] [SERVER]    [C1] Client connected from 127.0.0.1:50884
[19:15:03.242505600] [SERVER]    [C1] Client disconnected
```

Test A3

```
[19:16:04.019268900] [SYSTEM]    Which type of server do you wish to run?
[19:16:04.039477000] [SYSTEM]    [1] Iterative
[19:16:04.039477000] [SYSTEM]    [2] Concurrent
[19:16:04.039477000] [SYSTEM]    [3] HTTP
                                                >>> 1

[19:16:05.568304000] [SERVER]    Iterative server listening on port 4628
[19:16:11.052432400] [SERVER]    [C1] Client connected from 127.0.0.1:51447
[19:18:34.228025600] [SERVER]    [C1] Client disconnected
[19:18:34.229023100] [SERVER]    [C2] Client connected from 127.0.0.1:51501
```

Test B1

```
[19:19:46.766699900] [SYSTEM]    Which type of server do you wish to run?
[19:19:46.780663800] [SYSTEM]    [1] Iterative
[19:19:46.780663800] [SYSTEM]    [2] Concurrent
[19:19:46.780663800] [SYSTEM]    [3] HTTP
                                                >>> 2

[19:19:54.137580400] [SERVER]    Concurrent server listening on port 4628
[19:19:57.877015400] [SERVER]    [C1] Client connected from 127.0.0.1:52077
[19:19:57.888981800] [SERVER]    [C1] Starting worker thread
[19:19:57.889946800] [WORKER]    [C1] Worker thread started
```

Test B2

```
[19:19:46.766699900] [SYSTEM]    Which type of server do you wish to run?
[19:19:46.780663800] [SYSTEM]    [1] Iterative
[19:19:46.780663800] [SYSTEM]    [2] Concurrent
[19:19:46.780663800] [SYSTEM]    [3] HTTP
                                                >>> 2

[19:19:54.137580400] [SERVER]    Concurrent server listening on port 4628
[19:19:57.877015400] [SERVER]    [C1] Client connected from 127.0.0.1:52077
[19:19:57.888981800] [SERVER]    [C1] Starting worker thread
[19:19:57.889946800] [WORKER]    [C1] Worker thread started
[19:21:27.273516800] [SERVER]    [C1] Client disconnected
```

Test B3

```
[19:21:53.146099900] [SYSTEM]    Which type of server do you wish to run?
[19:21:53.162056600] [SYSTEM]    [1] Iterative
[19:21:53.162056600] [SYSTEM]    [2] Concurrent
[19:21:53.162056600] [SYSTEM]    [3] HTTP
                                                >>> 2

[19:22:25.097145800] [SERVER]    Concurrent server listening on port 4628
[19:22:27.548897800] [SERVER]    [C1] Client connected from 127.0.0.1:52495
[19:22:27.549951600] [SERVER]    [C1] Starting worker thread
[19:22:27.549951600] [WORKER]    [C1] Worker thread started
[19:22:29.491854300] [SERVER]    [C2] Client connected from 127.0.0.1:52501
[19:22:29.491854300] [SERVER]    [C2] Starting worker thread
[19:22:29.492415700] [WORKER]    [C2] Worker thread started
```

Test B4

```
[19:21:53.146099900] [SYSTEM]    Which type of server do you wish to run?
[19:21:53.162056600] [SYSTEM]    [1] Iterative
[19:21:53.162056600] [SYSTEM]    [2] Concurrent
[19:21:53.162056600] [SYSTEM]    [3] HTTP
                                                >>> 2
```

```
[19:22:25.097145800] [SERVER]    Concurrent server listening on port 4628
[19:22:27.548897800] [SERVER]    [C1] Client connected from 127.0.0.1:52495
[19:22:27.549951600] [SERVER]    [C1] Starting worker thread
[19:22:27.549951600] [WORKER]    [C1] Worker thread started
[19:22:29.491854300] [SERVER]    [C2] Client connected from 127.0.0.1:52501
[19:22:29.491854300] [SERVER]    [C2] Starting worker thread
[19:22:29.492415700] [WORKER]    [C2] Worker thread started
[19:24:00.856140700] [SERVER]    [C2] Client disconnected
[19:24:10.648056100] [WORKER]    [C1R1] +:50.0:50.0 (50.0 + 50.0) Result: 100.0
```

Test C1

```
[19:26:25.818096900] [WORKER]    [C1R1] +:127.0:16.0 (127.0 + 16.0) Result: 143.0
```

Test C2

```
[19:26:37.812416900] [WORKER]    [C1R2] -:743.0:287.0 (743.0 - 287.0) Result:
456.0
```

Test C3

```
[19:26:43.612549200] [WORKER]    [C1R3] *:41.0:59.0 (41.0 * 59.0) Result: 2419.0
```

Test C4

```
[19:26:55.986612400] [WORKER]    [C1R4] /:540.0:90.0 (540.0 / 90.0) Result: 6.0
```

Test C5

```
[19:27:06.309116500] [WORKER]    [C1R5] ^:2.0:8.0 (2.0 ^ 8.0) Result: 256.0
```

Test D1

```
[19:37:27.794738300] [CLIENT]    Do you want to do another calculation? y/n
                                                >>> y
[19:39:07.891033900] [CLIENT]    Please enter a calculation eg. 89 - 36.5
```

Test D2

```
[19:40:02.308794400] [CLIENT]    Do you want to do another calculation? y/n
                                                >>> n
[19:40:06.811965600] [CLIENT]    Connection closed
[19:40:06.811965600] [CLIENT]    Client closing

Process finished with exit code 1
```

Test D3

```
[19:41:37.661651500] [CLIENT]    Do you want to do another calculation? y/n
                                                >>> t
                                                >>>
```

Test D4

```
[19:42:09.997772100] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> a * 4
[19:42:27.692719300] [ERROR]     Alphabetical characters are not permitted
```

Test D5

```
[19:42:09.997772100] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> 500 *
[19:42:40.913492000] [ERROR]     Something's not quite right
```

Test D6

```
[19:43:34.001508000] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> 123 123
[19:43:38.142955100] [ERROR]     No valid operator found, valid operators include
'+', '-', '*', '/', '^'
```

Test D7

```
[19:47:14.099349900] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> 12 % 2
[19:47:22.853671300] [ERROR]     No valid operator found, valid operators include
'+', '-', '*', '/', '^'
```

Test D8

```
[19:47:48.414060300] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> 123 ++ 456
[19:47:55.753604300] [ERROR]     Equation invalid, please provide one operator
```

Test D9

```
[19:48:24.428967300] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> + 123 123
[19:48:28.934768600] [ERROR]     Something's not quite right
```

Discussion

Having basic client and server start-points, with menus to decide which to start; proved to be very helpful during the testing stages.

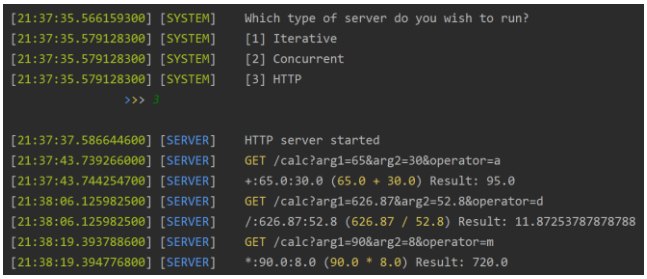
An error was identified during the testing of the calculations, it was discovered that negative number results were not being handled correctly by the SocketClient, this has since been fixed.

Assumptions

In creating the solutions to question 1; it has been assumed that all operands are positive real numbers, as per the requirement `<operator(+|-|*|/>:<operand1(x.x)>:<operand2(x.x)>`.

Question 2: HTTP-based Solution

Server

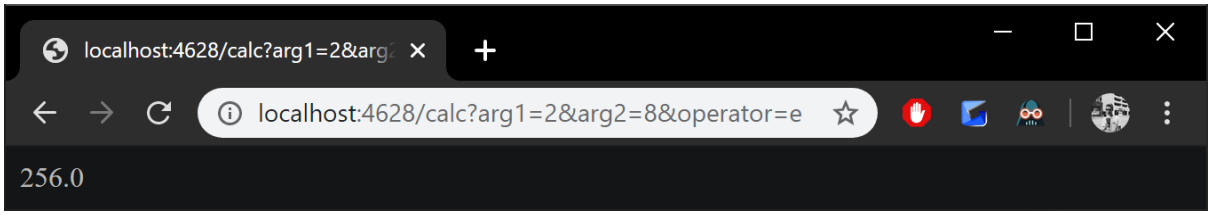


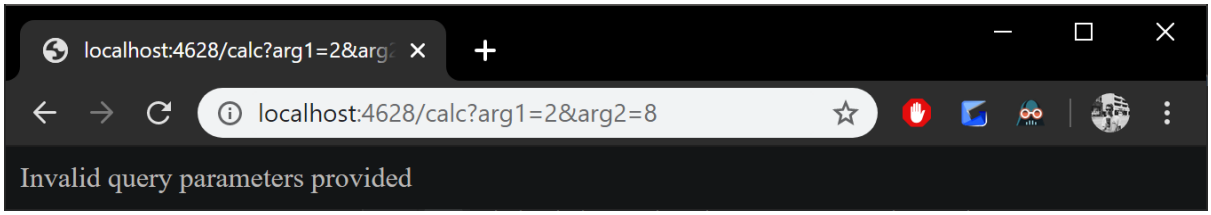
The HTTP server (appendix C) handles transactions from clients in a connectionless context. The server is set up to use the default executor and is provided with a context handler which deals with the request received by the client. The server instantiates an InetSocketAddress using a port which clients can send requests to (using `http://localhost:<PORT>/<CONTEXT>`). The HTTP server makes use of the MathService class in exactly the same way as it was used in the iterative and concurrent servers discussed in question 1. Calculation result is returned to the client via a `write()` method call on the request object.

Testing

This section provides a set of test cases used to verify the HTTP server using an internet browser. Full testing of the MathService was undertaken for question 1 and has not been repeated here.

No.	Description	Expected Result	Actual Result	Pass/Fail
1	Send a valid calculation request to the server to get 2 ^ 8.	"256" returned as text to the browser	As expected, see first screenshot after table	P
2	Send an invalid calculation request with a missing parameter (the operator).	"Invalid query parameters provided" error message	As expected, see second screenshot after table	P





Client

The HTTP client (appendix C) makes requests to the server using the following URL format `http://localhost:<PORT>/<CONTEXT>`. Similarly to the SocketClient, the HttpClient enables the user to make multiple requests to the server, using the yes/no prompt again. If the user chooses to make another calculation; then following the HTTP protocol, each calculation is handled as a completely separate request to the server (differing from the socket approach described in question 1).

Testing

User input validation tests were conducted as part of the test suite for the SocketClient in question 1. The code developed to perform validation of user input is all common to both the socket client and the HTTP client and therefore did not need to be re-tested here. The test cases specified in the table below are used to verify that calculations input by the user are correctly communicated to the server and the results are correctly communicated back to the client and displayed to the user.

No.	Description	Expected Result	Actual Result	Pass/Fail
1	Valid calculation using + symbol 31 + 76	Result = 107	As expected	P
2	Valid calculation using / symbol 100/20	Result = 5	As expected	P
3	Valid calculation using * symbol 45 * 7	Result = 315	As expected	P

Test 1

Client

```
[22:16:36.770713200] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> 31 + 76
[22:16:46.228960000] [SERVER]    107.0
```

Server

```
[22:16:46.203016600] [SERVER]    GET /calc?arg1=31.0&arg2=76.0&operator=a
[22:16:46.207006000] [SERVER]    +:31.0:76.0 (31.0 + 76.0) Result: 107.0
```

Test 2

Client

```
[22:16:52.108738000] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> 100 / 20
[22:16:56.832197000] [SERVER]    5.0
```

Server

```
[22:16:56.831198100] [SERVER]    GET /calc?arg1=100.0&arg2=20.0&operator=d
[22:16:56.831198100] [SERVER]    /:100.0:20.0 (100.0 / 20.0) Result: 5.0
```

Test 3

Client

```
[22:16:58.472526900] [CLIENT]    Please enter a calculation eg. 89 - 36.5
                                                >>> 45 * 7
[22:17:03.399179600] [SERVER]    315.0
```

Server

```
[22:17:03.398183300] [SERVER]    GET /calc?arg1=45.0&arg2=7.0&operator=m
[22:17:03.398183300] [SERVER]    *:45.0:7.0 (45.0 * 7.0) Result: 315.0
```