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

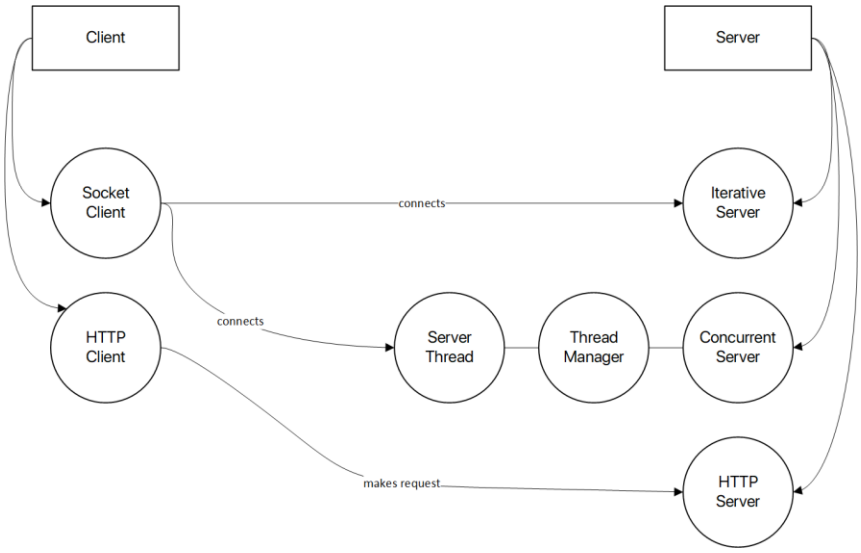


Figure 1: Overall Architecture

client. The overall architecture is shown in the following diagram (figure 1).

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. In total, the solutions for the three sets of client-server system comprise 22 classes. The project structure across all client and server types implemented in this assignment is given below.

```
└──mathapp
   │   Client.java
   │   Server.java
   │
   ├──common
   │       ClientBase.java
   │       Colors.java
   │       Constants.java
   │       Logger.java
   │       MathService.java
   │       Params.java
   │       Response.java
   │       ResponseType.java
   │       ServerBase.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. For the design of the iterative and concurrent solutions; UML sequence diagrams provide the most appropriate design approach and have been used later in this section.

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).

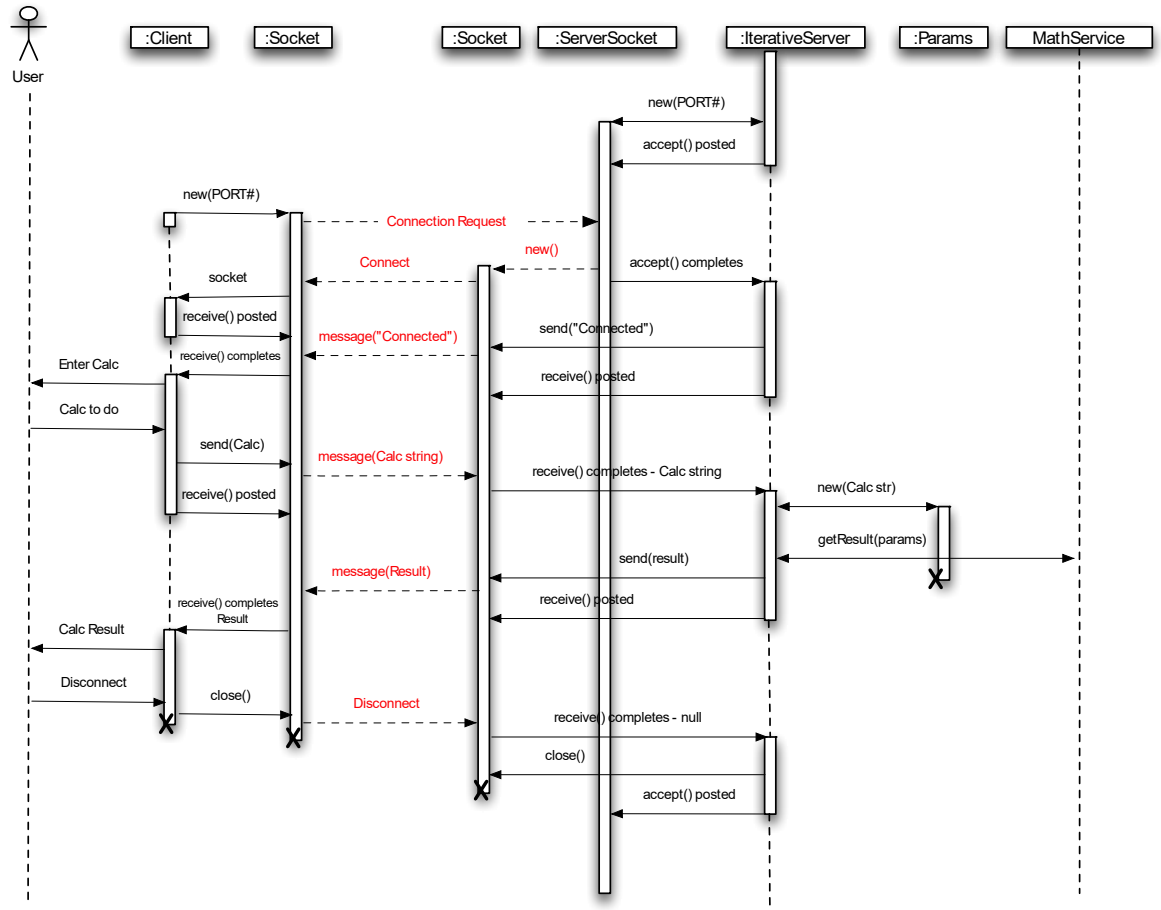


Figure 2: Iterative Server UML Sequence Diagram

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 (appendix B.2) 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.

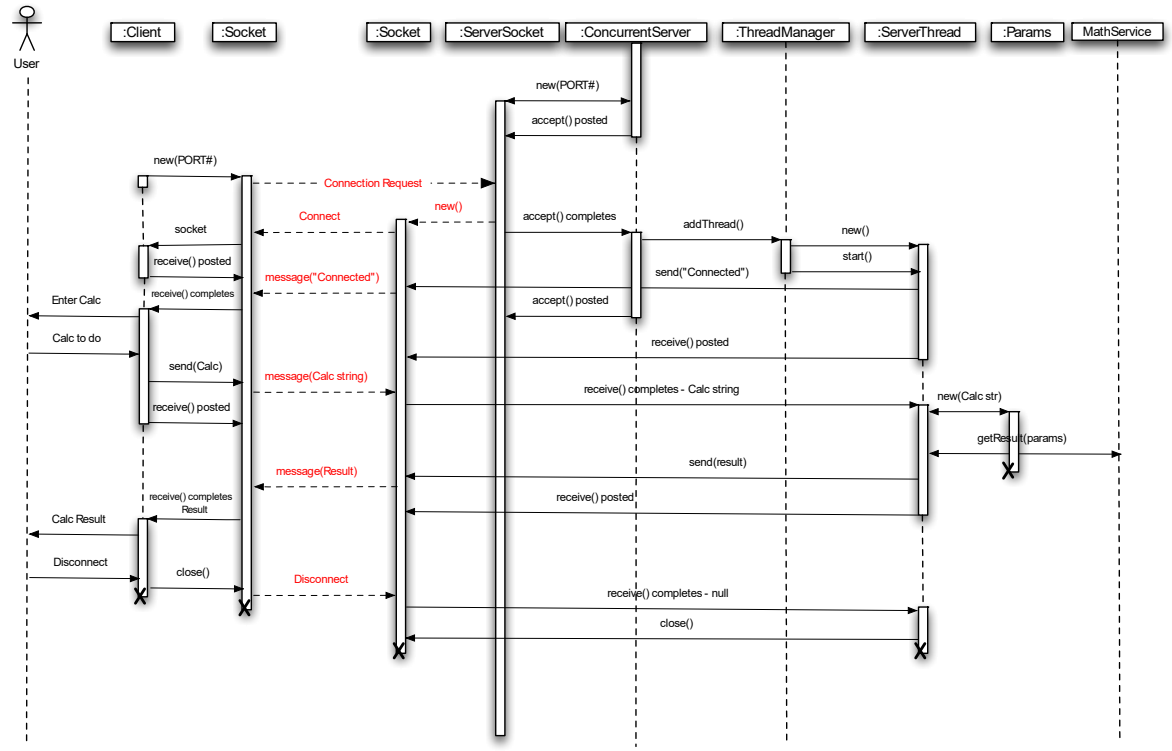


Figure 3: Concurrent Server UML Sequence Diagram

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.

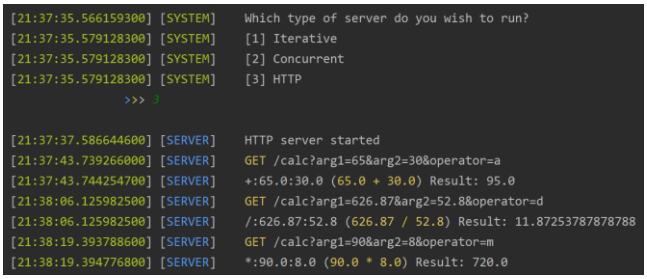
Assumptions

It has been assumed that the user is not constrained to enter the calculation in the same format by which it is communicated from the client to the server. A more natural scheme for the user to enter calculations has been adopted (such as `90 + 5`) and extensive validation is performed on the inputted data.

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

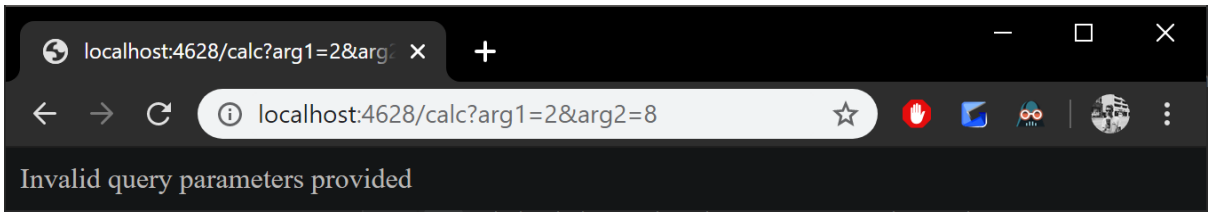


The HTTP server (appendix C) handles transactions from clients in a connectionless context. The server is set up to use the default executor and is provided with a context handler which deals with the request received by the client. The server instantiates an InetSocketAddress using a port which clients can send requests to (using `http://localhost:<PORT>/<CONTEXT>`). The HTTP server expects requests made to `/calc` to have three query parameters; `operator`, `arg1` and `arg2`. 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
```

Code Listing

Appendix A

MathApp root classes and all common classes used to support each of the solutions are listed in this appendix (11 classes total).

Client.java

```
package mathapp;

import java.util.Scanner;
import mathapp.common.Constants;
import mathapp.common.Logger;
import mathapp.http.client.HttpClient;
import mathapp.socket.client.SocketClient;

// This is the entry-point for client execution, here a decision is made on which type of
client to run

public class Client {

    public static void main(String[] args) {
        boolean acceptedValue = false;
        String input;

        System.out.println(Constants.APP_TITLE);

        Scanner scanner = new Scanner(System.in);

        Logger.system("Which type of client do you wish to run?");
        Logger.system("[1] Socket");
        Logger.system("[2] HTTP");

        while (!acceptedValue) {
            Logger.input();
            input = scanner.nextLine();
            if (input.length() > 0) {
                switch (input.substring(0, 1)) {
                    case "1":
                        acceptedValue = true;
                        new SocketClient();
                        break;
                    case "2":
                        acceptedValue = true;
                        new HttpClient();
                        break;
                    default:
                        acceptedValue = false;
                        break;
                }
            }
        }

        Logger.blank();
    }
}
```

Server.java

```
package mathapp;

import mathapp.common.Constants;
import mathapp.common.Logger;
import mathapp.common.ServerBase;
import mathapp.http.server.HTTPServer;
import mathapp.socket.server.iterative.IterativeServer;
import mathapp.socket.server.concurrent.ConcurrentServer;

import java.util.Scanner;

// This is the entry-point for server execution, here a decision is made on which type of
server to run

public class Server {
    private static ServerBase server;

    public static void main(String[] args) {
        boolean acceptedValue = false;
        String input;

        Scanner scanner = new Scanner(System.in);

        System.out.println(Constants.APP_TITLE);

        Logger.system("Which type of server do you wish to run?");
        Logger.system("[1] Iterative");
        Logger.system("[2] Concurrent");
        Logger.system("[3] HTTP");

        while (!acceptedValue) {
            Logger.input();
            input = scanner.nextLine();
            if (input.length() > 0) {
                switch (input.substring(0, 1)) {
                    case "1":
                        setServer(new IterativeServer());
                        break;
                    case "2":
                        setServer(new ConcurrentServer());
                        break;
                    case "3":
                        setServer(new HTTPServer());
                        break;
                    default:
                        server = null;
                        break;
                }
            }

            if (server != null) {
                acceptedValue = true;
            }
        }

        Logger.blank();
        server.start();
```

```
    }

    private static void setServer(Object serverObj) {
        try {
            server = (ServerBase) serverObj;
        } catch (Exception ex) {
            server = null;
            Logger.error(ex);
        }
    }
}
```

Colors.java

```
package mathapp.common;

// This class collect together all the constant definitions used in
// printing text in a variety of colours

public class Colors {
    public static final String ANSI_RESET = "\u001B[0m";
    public static final String ANSI_BLACK = "\u001B[30m";
    public static final String ANSI_RED = "\u001B[31m";
    public static final String ANSI_YELLOW = "\u001B[33m";
    public static final String ANSI_GREEN = "\u001B[32m";
    public static final String ANSI_BLUE = "\u001B[34m";
    public static final String ANSI_PURPLE = "\u001B[35m";
    public static final String ANSI_CYAN = "\u001B[36m";
    public static final String ANSI_WHITE = "\u001B[37m";
}
```

Constants.java

```
package mathapp.common;

// Provides storage for commonly used values

public class Constants {
    public static final int PORT = 4628;
    public static final String BASE_URI = "http://localhost:" + PORT + "/calc";

    public static final String APP_TITLE= " __  __       _   _                                \n"
        + "|  \\/  |     | | | |       /\\                    \n"
        + "| \\  / | __ _| |_| |__    /  \\   _ __  _ __    \n"
        + "| |\\/| |/ _` | __| '_ \\  / /\\ \\ | '_ \\| '_ \\ \n"
        + "| |  | | (_| | |_| | | |/ ____ \\| |_) | |_) |\n"
        + "|_|  |_|\\__,_|\\__|_| |_/_/    \\_\\ .__/| .__/ \n"
        + "                                 | |   | |    \n"
        + "                                 |_|   |_|    \n";
}
```

ClientBase.java

```
package mathapp.common;

import java.io.BufferedReader;

// This class contain generic methods for managing and validating user input

public class ClientBase {

    // Method attempts to obtain a valid calculation command from the user
    protected static Params getValidInput(BufferedReader input) {
        Params params = null;
        String test, permittedOperators = "+-*/^";
        String[] testElements;
        double arg1, arg2;
        boolean error;
        int operatorIndex;

        Logger.client("Please enter a calculation eg. 89 - 36.5");

        while (params == null) {
            error = false;
            operatorIndex = -1;

            Logger.input();

            try {
                test = input.readLine().trim().replaceAll(" +", "");

                // Validation
                // If input contains characters
                if (test.matches(".*[a-zA-Z]+.*")) {
                    Logger.error("Alphabetical characters are not permitted");
                    continue;
                }

                // Find index of operator
                for (int i = 0; i < test.length(); i++) {
                    for (char c : permittedOperators.toCharArray()) {
                        if (test.charAt(i) == c) {
                            if (operatorIndex == -1) {
                                operatorIndex = i;
                            } else {
                                Logger.error("Equation invalid, please provide one operator");
                                error = true;
                                break;
                            }
                        }
                    }
                    if (error || (i == test.length() - 1 && operatorIndex == -1)) {
                        if (!error) {
                            Logger.error("No valid operator found, valid operators include '+',
'-', '*', '/', '^'");
                            error = true;
                        }
                        break;
                    }
                }
```

```
                if (error) {
                    continue;
                }

                if (operatorIndex != -1) {
                    if (operatorIndex == 0 || operatorIndex == test.length() - 1) {
                        Logger.error("Something's not quite right");
                        continue;
                    }

                    if (test.charAt(operatorIndex + 1) != ' ') {
                        test = insertString(test, " ", operatorIndex + 1);
                    }
                    if (test.charAt(operatorIndex - 1) != ' ') {
                        test = insertString(test, " ", operatorIndex);
                    }
                } else {
                    Logger.error("No valid operator found, valid operators include '+', '-',
'*', '/', '^'");
                    continue;
                }

                testElements = test.split(" ");
                if (testElements.length == 3) {
                    arg1 = Double.parseDouble(testElements[0]);
                    arg2 = Double.parseDouble(testElements[2]);

                    params = new Params(testElements[1], arg1, arg2);
                }

            } catch (Exception ex) {
                Logger.error(ex);
                params = null;
            }
        }

        return params;
    }

    // Method attempts to obtain a yes/no response from user
    protected static boolean getYesNo(BufferedReader input, String message) {
        boolean valueAcquired = false, value = false;
        Logger.client(message + " y/n");

        while (!valueAcquired) {
            Logger.input();
            try {
                switch (input.readLine().toLowerCase().charAt(0)) {
                    case 'y':
                        value = true;
                        valueAcquired = true;
                        break;
                    case 'n':
                        value = false;
                        valueAcquired = true;
                        break;
                    default:
                        valueAcquired = false;
                        break;
```

```
                }
            } catch (Exception ex) {
                Logger.error(ex);
            }
        }

        return value;
    }

    private static String insertString(String originalString, String stringToBeInserted, int
index) {
        return new StringBuilder(originalString).insert(index, stringToBeInserted).toString();
    }
}
```

Logger.java

```
package mathapp.common;

import java.time.LocalTime;

// This class contains a range of static methods used for logging out data to the console

public class Logger {

    private static String log(LogType type, String message) {
        String color = Colors.ANSI_RESET;
        String _type = type.name();
        String padding = "";
        int paddingLength = 10 - _type.length();

        for (int i = 0; i < paddingLength; i++)
            padding += ' ';

        switch (type) {
            case SYSTEM:
                color = Colors.ANSI_GREEN;
                break;
            case SERVER:
                color = Colors.ANSI_BLUE;
                break;
            case CLIENT:
                color = Colors.ANSI_YELLOW;
                break;
            case WORKER:
                color = Colors.ANSI_PURPLE;
                break;
            case ERROR:
                color = Colors.ANSI_RED;
                break;
        }

        _type = Colors.ANSI_RESET + "[" + color + _type + Colors.ANSI_RESET + "]";
        String currentTime = LocalTime.now().toString();
        while (currentTime.length() != 18)
            currentTime = currentTime.concat("0");

        String time = "[" + Colors.ANSI_GREEN + currentTime + Colors.ANSI_RESET + "] ";

        return time + _type + padding + message + Colors.ANSI_RESET;
```

```
    }

    private static void print(String message, boolean line) {
        if (line) {
            System.out.println(message);
        } else {
            System.out.print(message);
        }
    }

    public static void blank() {
        System.out.println();
    }

    public static void input() {
        System.out.print("\t\t\t\t\t\t\t  " + Colors.ANSI_BLUE + ">" + Colors.ANSI_YELLOW + ">"
+ Colors.ANSI_RESET + "> ");
    }

    public static void system(String message) {
        system(message, true);
    }

    public static void system(String message, boolean line) {
        print(log(LogType.SYSTEM, message), line);
    }

    public static void server(String message) {
        server(message, true);
    }

    public static void server(String message, boolean line) {
        print(log(LogType.SERVER, message), line);
    }

    public static void worker(String message) {
        worker(message, true);
    }

    public static void worker(String message, boolean line) {
        print(log(LogType.WORKER, message), line);
    }

    public static void client(String message) {
        client(message, true);
    }

    public static void client(String message, boolean line) {
        print(log(LogType.CLIENT, message), line);
    }

    public static void error(String message) {
        print(log(LogType.ERROR, message), true);
    }

    public static void error(Exception ex) {
        String msg = ex.getMessage();
        try {
            msg = msg.substring(0, 1).toUpperCase() + msg.substring(1);
```

```
        } catch (Exception e) {
            msg = "An error occurred";
        }
        print(log(LogType.ERROR, msg), true);
    }

    public static String formatId(String value) {
        return "[" + Colors.ANSI_BLUE + value + Colors.ANSI_RESET + "] ";
    }
}

enum LogType {SYSTEM, SERVER, WORKER, CLIENT, ERROR}
```

ResponseType.java

```
package mathapp.common;

public enum ResponseType {RESULT, MESSAGE, ERROR}
```

MathService.java

```java
package mathapp.common;

// MathService is used by all three server types, the getResult method is supplied with a
Params
// object and it returns the result as a string

public class MathService {
    private static double add(double a, double b) {
        return a + b;
    }
    private static double sub(double a, double b) {
        return a - b;
    }
    private static double mul(double a, double b) {
        return a * b;
    }
    private static double div(double a, double b) {
        return a / b;
    }
    private static double exp(double a, double b) {
        try {
            return Math.pow(a, b);
        } catch (Exception e) {
            System.out.println(e.getClass().getName());
            return 0;
        }
    }

    public static String getResult(Params params) {
        double result;
        double[] args = params.getArgs();
        switch (params.getOperator()) {
            case "+":
                result = MathService.add(args[0], args[1]);
                break;
            case "-":
                result = MathService.sub(args[0], args[1]);
                break;
            case "*":
                result = MathService.mul(args[0], args[1]);
                break;
            case "/":
                result = MathService.div(args[0], args[1]);
                break;
            case "^":
                result = MathService.exp(args[0], args[1]);
                break;
            default:
                return "";
        }
        return Double.toString(result);
    }
}
```

Params.java

```
package mathapp.common;

import java.util.Map;

// This class manages parameters for the maths calculation, involving one operator and two
arguments,
// it builds builds the calculation string in the format required to be communicated from
client
// and server

public class Params {

    private String operator;
    private double arg1, arg2;

    Params(String operator, double arg1, double arg2) {
        this.operator = operator;
        this.arg1 = arg1;
        this.arg2 = arg2;
    }

    public String getOperator() {
        return operator;
    }

    public double[] getArgs() {
        double[] args = new double[2];
        args[0] = arg1;
        args[1] = arg2;
        return args;
    }

    // Creates the calculation as a string in the format required by the server
    public String buildString() {
        return String.join(":", operator, Double.toString(arg1), Double.toString(arg2));
    }

    // Creates the calculation as a query string required by the HTTP server
    public String toQueryString() {
        String safeOperator;
        switch (this.operator) {
            default:
            case "+":
                safeOperator = "a";
                break;
            case "-":
                safeOperator = "s";
                break;
            case "*":
                safeOperator = "m";
                break;
            case "/":
                safeOperator = "d";
                break;
            case "^":
                safeOperator = "e";
                break;
```

```
        }
        return "?arg1=" + arg1 + "&arg2=" + arg2 + "&operator=" + safeOperator;
    }

    // Presents calculation in a human-readable format
    @Override
    public String toString() {
        return Colors.ANSI_YELLOW + String.join(" " + operator + " ", Double.toString(arg1),
Double.toString(arg2)) + Colors.ANSI_RESET;
    }

    // Method decomposes received string by the server into a Params object
    public static Params fromString(String value) throws IllegalArgumentException {
        try {
            String[] params = value.split(":");
            if (params.length != 3) {
                throw new Exception();
            }
            return new Params(params[0], Double.parseDouble(params[1]),
Double.parseDouble(params[2]));
        } catch (Exception ex) {
            throw new IllegalArgumentException("Value: " + value + " Error" + ex.getMessage());
        }
    }

    // Method decomposes a map of query string parameters into a Params object
    public static Params fromQueryString(Map<String, String> queryParameters) throws
IllegalArgumentException {
        String operatorValue;
        double value1, value2;
        try {
            operatorValue = queryParameters.get("operator");
            value1 = Double.parseDouble(queryParameters.get("arg1"));
            value2 = Double.parseDouble(queryParameters.get("arg2"));

            if (operatorValue.length() > 0) {
                switch (operatorValue.substring(0, 1)) {
                    case "a":
                        operatorValue = "+";
                        break;
                    case "s":
                        operatorValue = "-";
                        break;
                    case "m":
                        operatorValue = "*";
                        break;
                    case "d":
                        operatorValue = "/";
                        break;
                    case "e":
                        operatorValue = "^";
                        break;
                    default:
                        throw new Exception();
                }
            } else {
                throw new Exception();
            }
```

```
            return new Params(operatorValue, value1, value2);

        } catch (Exception ex) {
            throw new IllegalArgumentException("Invalid query parameters provided");
        }
    }
}
```

ServerBase.java

```
package mathapp.common;

// This interface is being used to ensure all three server types (iterative, concurrent and
HTTP)
// can be treated equally by mathapp.Server

public interface ServerBase {
    void start();
}
```

Response.java

```
package mathapp.common;

// This class is concerned with logging, fromString determines which type of log should
// be printed depending on the response from the server

public class Response {
    private ResponseType type;

    private Response(String type, String message) {
        switch (type) {
            case "ERROR":
                Logger.error(message);
                this.type = ResponseType.ERROR;
                break;
            case "RESULT":
                Logger.server("Result: " + message);
                this.type = ResponseType.RESULT;
                break;
            case "MESSAGE":
            default:
                Logger.server(message);
                this.type = ResponseType.MESSAGE;
                break;
        }
    }

    public ResponseType getType() {
        return type;
    }

    public static Response fromString(String data) throws Exception {
        try {
            String[] responseElements = data.split("#");
            return new Response(responseElements[0], responseElements[1]);
        } catch (Exception ex) {
            throw new Exception("Invalid response from server");
        }
    }
}
```

Appendix B

The following four classes are used across the socket-based client-server solutions.

IOSocket.java

```
package mathapp.socket;

import mathapp.common.Colors;
import mathapp.common.Logger;
import mathapp.common.ResponseType;

import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;

// Used by both client and server, wraps a java.net.Socket object and adds send()
// and receive() methods for communication

public class IOSocket {
    private Socket socket;
    private BufferedReader input;
    private PrintWriter output;

    public IOSocket(Socket socket) throws IOException {
        this.socket = socket;
        this.initialise();
    }

    public void close() {
        try {
            this.socket.close();
        } catch (Exception ex) {
            Logger.error(ex);
        }
    }

    private void initialise() throws IOException {
        // Get an input stream for reading character-mode input (BufferedReader)
        this.input = new BufferedReader(new InputStreamReader(this.socket.getInputStream()));

        // Get an output stream  for writing character-mode output (PrintWriter)
        this.output = new PrintWriter(new OutputStreamWriter(this.socket.getOutputStream()));
    }

    public String getIpAddress() {
        return Colors.ANSI_GREEN + this.socket.getInetAddress().toString().replace('/', '
').trim() + ":" + this.socket.getPort() + Colors.ANSI_RESET;
    }

    public void send(String message) throws IOException {
        output.println(message);
        // The ensuing flush method call is necessary for the data to
        // be written to the socket data stream before the socket is closed.
        output.flush();
    }
```

```
    // Sends a message across the socket
    public void send(ResponseType type, String message) throws IOException {
        this.send(String.join("#", type.name(), message));
    }

    // Receives a message across the socket
    public String receive() throws IOException {
        // read a line from the data stream
        return input.readLine();
    }
}
```

Request.java

```
package mathapp.socket.server;

import mathapp.common.Params;

// The Request class is used for logging purposes

public class Request {
    private String id;
    private Params params;
    private String result;

    public String getId() {
        return this.id;
    }

    Request(ServerConnection connection, Params params, int number, String result) {
        this.id = connection.getId() + "R" + number;
        this.params = params;
        this.result = result;
    }
}
```

ServerConnectionLog.java

```
package mathapp.socket.server;

import java.util.HashMap;

// This class is only used to keep a track of previous connections

public class ServerConnectionLog {
    private HashMap<String, ServerConnection> log;

    public ServerConnectionLog() {
        this.log = new HashMap<>();
    }

    void addItem(String id, ServerConnection connection) {
        this.log.put(id, connection);
    }
}
```

ServerConnection.java

```java
package mathapp.socket.server;

import mathapp.common.Params;
import mathapp.socket.IOSocket;

import java.io.IOException;
import java.net.Socket;
import java.util.ArrayList;

// This class is used for managing the server's IOSocket, and also handles logging

public class ServerConnection {
    private IOSocket socket;
    private String id;
    private ServerConnectionLog log;
    private ArrayList<Request> requests;

    public ServerConnection(Socket socket, int number, ServerConnectionLog log) throws
IOException {
        this.socket = new IOSocket(socket);
        this.id = "C" + number;
        this.log = log;
        this.requests = new ArrayList<>();

        log.addItem(id, this);
    }

    public IOSocket getSocket() {
        return this.socket;
    }

    public String getId() {
        return this.id;
    }

    public String getIpAddress() {
        return this.socket.getIpAddress();
    }

    public ArrayList<Request> getRequests() {
        return this.requests;
    }

    public Request addRequest(Params params, int number, String result) {
        // this method is used to maintain the ServerConnectionLog which is given as a
parameter to the constructor

        Request request = new Request(this, params, number, result);
        this.requests.add(request);
        this.log.addItem(this.id, this);
        return request;
    }
}
```

Appendix B.1 – Iterative Server Solution

IterativeServer.java

```
package mathapp.socket.server.iterative;

import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

import mathapp.common.ServerBase;
import mathapp.common.*;
import mathapp.socket.server.Request;
import mathapp.socket.server.ServerConnection;
import mathapp.socket.server.ServerConnectionLog;

// This class handles the connection request and the transaction involved in the call from a
client

public class IterativeServer implements ServerBase {

    // A boolean flag to control the while loop that handles connections and their requests
    private boolean running;

    // Integer values used for generating ID's for connections/requests
    private int connectionCount, requestCount;

    private ServerConnectionLog log;

    public IterativeServer() {
        this.running = true;
        this.connectionCount = 0;
        this.requestCount = 0;
        this.log = new ServerConnectionLog();
    }

    // Called from mathapp.Server
    public void start() {
        Socket client;
        String data;

        try {
            // Establishes port for clients to connect through
            ServerSocket serverSocket = new ServerSocket(Constants.PORT);
            Logger.server("Iterative server listening on port " + Colors.ANSI_YELLOW +
Constants.PORT + Colors.ANSI_RESET);

            ServerConnection connection;
            Request request;

            while (this.running) {
                try {
                    // Waits for client to connect to server
                    client = serverSocket.accept();
                    this.connectionCount++;
                    this.requestCount = 0;

                    connection = new ServerConnection(client, this.connectionCount, this.log);
```

```
                    try {
                        Logger.server(Logger.formatId(connection.getId()) + "Client connected
from " + connection.getIpAddress());
                        connection.getSocket().send(ResponseType.MESSAGE, "Connected");

                        Params params;
                        String result;

                        // While client is connected
                        while ((data = connection.getSocket().receive()) != null) {
                            try {
                                // This block gets the parameters for the calculation from the
client, performs
                                // the necessary calculation and returns the necessary result
back to the client

                                this.requestCount++;
                                params = Params.fromString(data);
                                result = MathService.getResult(params);

                                request = connection.addRequest(params, this.requestCount,
result);
                                Logger.server(Logger.formatId(request.getId()) +
params.buildString() + " (" + params.toString() + ") Result: " + result);
                                connection.getSocket().send(ResponseType.RESULT, result);

                            } catch (Exception ex) {
                                if (ex.getClass() == SocketException.class) {
                                    Logger.server(Logger.formatId(connection.getId()) + "Client
disconnected");
                                } else {
                                    Logger.error(ex);
                                }
                            }
                        }

                        // At this point the client has disconnected from the server so the
client's
                        // connection will be closed and the server will loop back round
waiting for
                        // another client to connect

                        Logger.server(Logger.formatId(connection.getId()) + "Client
disconnected");
                        client.close();
                    } catch (Exception ex) {
                        if (ex.getClass() == SocketException.class) {
                            Logger.server(Logger.formatId(connection.getId()) + "Client
disconnected");
                        } else {
                            Logger.error(ex);
                        }
                    }

                } catch (Exception ex) {
                    ex.printStackTrace();
                    Logger.error(ex);
                    if (ex.getClass() != SocketException.class) {
                        Logger.server(Colors.ANSI_RED + ex.getMessage() + Colors.ANSI_RESET + "
```

```
" + ex.getClass().getTypeName());
                        Logger.system("Exiting");
                    }
                }
            }
        } catch (Exception ex) {
            Logger.error(ex);
        }
    }
}
```

Appendix B.2 – Concurrent Server Solution

ConcurrentServer.java

```
package mathapp.socket.server.concurrent;

import mathapp.common.ServerBase;
import mathapp.common.Logger;
import mathapp.common.Colors;
import mathapp.common.Constants;
import mathapp.socket.server.ServerConnection;
import mathapp.socket.server.ServerConnectionLog;

import java.net.ServerSocket;
import java.net.Socket;

// This class handles the connection request and starts a ServerThread for each connection

public class ConcurrentServer implements ServerBase {

    private boolean running;
    private int connectionCount;
    private ServerConnectionLog log;
    private ThreadManager threadManager;

    public ConcurrentServer() {
        this.running = true;
        this.connectionCount = 0;
        this.log = new ServerConnectionLog();
        this.threadManager = new ThreadManager();
    }

    public void start() {
        ServerConnection connection;
        Socket client;

        try {
            // Establishes port for clients to connect through
            ServerSocket serverSocket = new ServerSocket(Constants.PORT);

            Logger.server("Concurrent server listening on port " + Colors.ANSI_YELLOW +
Constants.PORT + Colors.ANSI_RESET);

            while (this.running) {
                try {
                    // Waits for client to connect to server
                    client = serverSocket.accept();
                    this.connectionCount++;

                    this.threadManager.closeCompleted();

                    connection = new ServerConnection(client, this.connectionCount, this.log);
                    Logger.server(Logger.formatId(connection.getId()) + "Client connected from
" + connection.getIpAddress());

                    this.threadManager.addThread(new ServerThread(connection));
                } catch (Exception ex) {
                    Logger.error(ex);
                }
```

```
            }
        } catch (Exception ex) {
            Logger.error(ex);
        }
    }
}
```

ServerThread.java

```
package mathapp.socket.server.concurrent;

import java.net.SocketException;

import mathapp.common.Logger;
import mathapp.common.MathService;
import mathapp.common.Params;
import mathapp.common.ResponseType;
import mathapp.socket.server.Request;
import mathapp.socket.server.ServerConnection;

// An instance of this class is created to service each client connection

public class ServerThread extends Thread {

    private ServerConnection connection;

    ServerThread(ServerConnection connection) {
        this.connection = connection;
    }

    ServerConnection getConnection() {
        return this.connection;
    }

    @Override
    public void run() {
        int requestCount = 0;
        String data;
        Request request;

        try {
            Logger.worker(Logger.formatId(this.connection.getId()) + "Worker thread started");

            Params params;
            String result;

            this.connection.getSocket().send(ResponseType.MESSAGE, "Connected");

            // While client is connected
            while ((data = this.connection.getSocket().receive()) != null) {
                try {
                    // This block gets the parameters for the calculation from the client,
performs
                    // the necessary calculation and returns the necessary result back to the
client

                    requestCount++;
                    params = Params.fromString(data);
```

```
                    result = MathService.getResult(params);

                    request = this.connection.addRequest(params, requestCount, result);
                    Logger.worker(
                        Logger.formatId(request.getId()) + params.buildString() + " (" + params
                            .toString() + ") Result: " + result);
                    this.connection.getSocket().send(ResponseType.RESULT, result);

                } catch (Exception ex) {
                    if (ex.getClass() == SocketException.class) {
                        break;
                    } else {
                        Logger.error(ex);
                    }
                }
            }

            Logger.server(Logger.formatId(this.connection.getId()) + "Client disconnected");

        } catch (Exception ex) {
            if (ex.getClass() == SocketException.class) {
                Logger.server(Logger.formatId(this.connection.getId()) + "Client
disconnected");
            } else {
                Logger.error(ex);
            }
        }
        this.interrupt();
    }
}
```

ThreadManager.java

```
package mathapp.socket.server.concurrent;

import mathapp.common.Logger;

import java.util.HashMap;
import java.util.Map.Entry;
import java.util.UUID;

// This class manages running ServerThreads

class ThreadManager {
    private HashMap<String, ServerThread> threads;

    ThreadManager() {
        this.threads = new HashMap<>();
    }

    // Adds a new ServerThread and starts it
    void addThread(ServerThread thread) {
        Logger.server(Logger.formatId(thread.getConnection().getId()) + "Starting worker
thread");
        thread.start();
        this.threads.put(UUID.randomUUID().toString().toUpperCase(), thread);
    }

    // Iterates over the map of ServerThreads and removes any which have been interrupted
    void closeCompleted() {
        for (Entry<String, ServerThread> threadItem : this.threads.entrySet()) {
            try {
                if (threadItem.getValue().isInterrupted()) {

Logger.server(Logger.formatId(threadItem.getValue().getConnection().getId()) + "Ending worker
thread");
                    this.threads.remove(threadItem.getKey());
                }
            } catch (Exception ex) {
                Logger.error(ex);
            }
        }
    }
}
```

Appendix B.3 – Socket-based Client Solution

SocketClient.java

```
package mathapp.socket.client;

import java.io.*;
import java.net.*;

import mathapp.common.*;
import mathapp.socket.IOSocket;

// This class provides the client for both the iterative and concurrent servers

public class SocketClient extends ClientBase {

    public SocketClient() {
        IOSocket socket;
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        Params params;
        String data;
        Response response;

        try {
            Logger.client("Attempting to connect to server on port " + Constants.PORT);

            // Instantiates a new IOSocket using a port number in common with the server it
wishes to connect to
            socket = new IOSocket(new Socket("localhost", Constants.PORT));

            currentConnection:
            while ((data = socket.receive()) != null) {
                // Loops while client is connected to server, allowing one or many calculation
requests

                try {
                    response = Response.fromString(data);
                    switch (response.getType()) {
                        case RESULT:
                            if (!getYesNo(input, "Do you want to do another calculation?")) {
                                break currentConnection;
                            }
                        default:
                            break;
                        case ERROR:
                            break currentConnection;
                    }

                    // Getting a valid Params object from the user input
                    params = getValidInput(input);

                    // Sending the command string to the server
                    socket.send(params.buildString());

                } catch (Exception ex) {
                    Logger.error(ex);
                }
            }
```

```
            input.close();
            socket.close();
            Logger.client("Connection closed");
        } catch (Exception ex) {
            Logger.error(ex);
        }

        Logger.client("Client closing");
        System.exit(1);
    }
}
```

Appendix C

HttpClient.java

```
package mathapp.http.client;

import java.io.*;

import org.apache.http.client.fluent.Request;

import mathapp.common.*;
import mathapp.common.ClientBase;

// This class provides the client for the HTTP server

public class HttpClient extends ClientBase {

    public HttpClient() {
        boolean running = true;
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        Params params;
        String data;

        while (running) {
            // Getting a valid Params object from the user input
            params = getValidInput(input);

            try {
                // Sending the command string to the server via a HTTP GET request
                data = Request.Get(Constants.BASE_URI + params.toQueryString())
                        .connectTimeout(1000)
                        .socketTimeout(1000)
                        .execute()
                        .returnContent()
                        .asString();

                Logger.server(data);
                if (!getYesNo(input, "Do you want to do another calculation?")) {
                    running = false;
                }
            } catch (Exception ex) {
                Logger.error(ex);
            }
        }
    }
}
```

HttpServer.java

```
package mathapp.http.server;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.util.HashMap;
import java.util.Map;
import mathapp.common.Colors;
import mathapp.common.Constants;
import mathapp.common.Logger;
import mathapp.common.MathService;
import mathapp.common.Params;
import mathapp.common.ServerBase;

// This class implements an HTTP server

public class HTTPServer implements ServerBase {

    public void start() {
        try {
            HttpServer server = HttpServer.create(new InetSocketAddress(Constants.PORT), 0);
            Logger.server("HTTP server started");
            server.createContext("/calc", new CalcContextHandler());
            server.setExecutor(null); // creates a default executor
            server.start();
        } catch (IOException ex) {
            Logger.error(ex);
        }
    }

    // Registered handler class for named context
    static class CalcContextHandler implements HttpHandler {

        @Override
        public void handle(HttpExchange request) throws IOException {
            Logger.server(
                Colors.ANSI_YELLOW + request.getRequestMethod() + Colors.ANSI_RESET + " " +
request
                    .getRequestURI().toString());
            //set to text/html for machine to machine communication
            request.getResponseHeaders().set("Content-Type", "text/html");

            String response = "";
            // Handle request type
            if (request.getRequestMethod().equalsIgnoreCase("GET")) {
                response = handleGET(request);
                if (response.equals("")) {
                    response = "Invalid query parameters provided";
                    request.sendResponseHeaders(400, 0); // 400 bad request
                } else {
                    request.sendResponseHeaders(200, 0); // 200 Ok
                }
            } else {
                request.sendResponseHeaders(501, 0); //  501 - not implemented
            }
```

```
            // Write response and close
            request.getResponseBody().write(response.getBytes());
            request.getResponseBody().close();
        }

        // Handle a HTTP GET request
        static String handleGET(HttpExchange request) throws NumberFormatException {
            Map<String, String> queryParameters = getQueryParameters(request);

            try {
                Params params = Params.fromQueryString(queryParameters);
                String result = MathService.getResult(params);
                Logger.server(
                    params.buildString() + " (" + params.toString() + ") Result: " + result);
                return result;
            } catch (Exception ex) {
                Logger.error(ex);
                return "";
            }
        }

        // Parse request query parameters into a map
        static Map<String, String> getQueryParameters(HttpExchange request) {
            Map<String, String> result = new HashMap<>();
            String query = request.getRequestURI().getQuery();
            if (query != null) {
                for (String param : query.split("&")) {
                    String pair[] = param.split("=");
                    if (pair.length > 1) {
                        result.put(pair[0], pair[1]);
                    } else {
                        result.put(pair[0], "");
                    }
                }
            }
            return result;
        }
    }
}
```