Middleware Architecture

(SCS 3103)

Mini Project

Implementation of a Middleware

Documentation

Name	Index No.
Kaveesha Baddage	14000091
Uthpala Pitawela	14001187
Viduni Wickramarachchi	14001543
Savinda Maddage	14000652

Table of Contents

1. Introduction

1.1 Purpose and Scope

Purpose

The purpose of the object oriented middleware which was developed is to enable a service to be registered and for the services provided by service providers to be utilized by the service consumers. Further, this middleware provides various services such as message passing and arithmetic operations. Successful communication between clients are ensured by the middleware and provides acknowledgements to verify successful communication as well.

Scope

The scope of the middleware is as follows.

- Register a service (server)
- Register Clients
- Lookup a service
- Successful communication between clients
- Acknowledgments
- Multiple Requests
- Error Handling when messages are not in the correct format and when the requested clients are not online

However, error handling of services, error handling when client and server failures occur is not managed by the middleware developed.

Prerequisites for users

The users of this middleware should have the basic knowledge of working on a terminal. All users using Windows, Linux or Mac operating systems have the ability of using this middleware.

1.2 Process Overview

The overall process managed by the middleware is as follows.

- Once the server is run, the services get registered.
- Once the client is run, each client gets registered with the server, as objects. These registered clients will be in the server's register.
- Each client can make either single requests or multiple requests from service providers.
- The clients should abide by a message format in order to request for services. If not and error is seen by the client on the terminal.
- The middleware component in the server monitors and enables the communication between clients.
- The server performs the operations and executes the requests by clients.
- Acknowledgments are received upon the successful execution of an operation.

2. Installation and Setup

Once the project is downloaded, open two terminals (one for a client, one for the server) and change the current working directory to the directory of the application.

```
cd /PATH/Middleware
```

Next, in order to compile the server, the following command should be run on one terminal (ideally, on the terminal for the server).

```
javac MultiThreadServer.java
```

Afterwards, in order to compile the client, the following command should be run on the other terminal.

```
javac MultiThreadChatClient.java
```

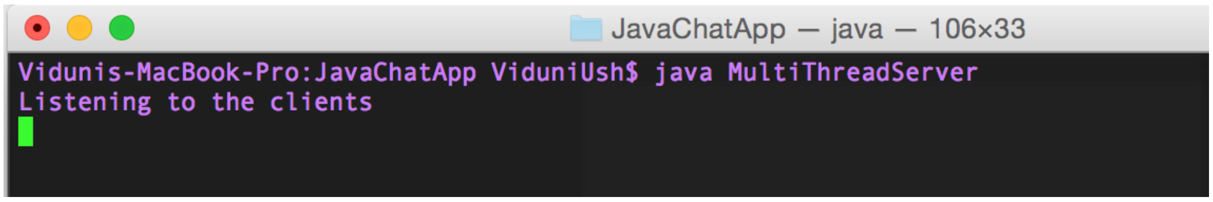
Next, on the server terminal, the following command should be run in order to run the server. As the server is run, the services get registered as well.

```
java MultiThreadServer
```

On the client terminal, the following command should be run in order to register the client with the server.

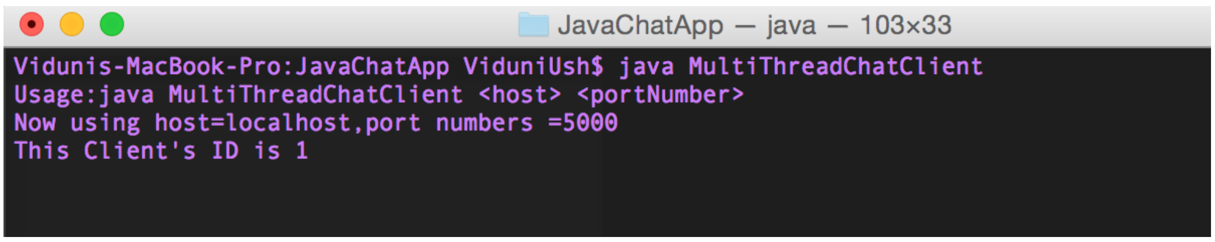
```
java MultiThreadChatClient
```

Using this command several clients can be registered with the server. Each registered client gets a client ID depending on the order they register with the hash map of the server. These client IDs are used by other service consumers to communicate with the relevant client.



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadServer
Listening to the clients
```

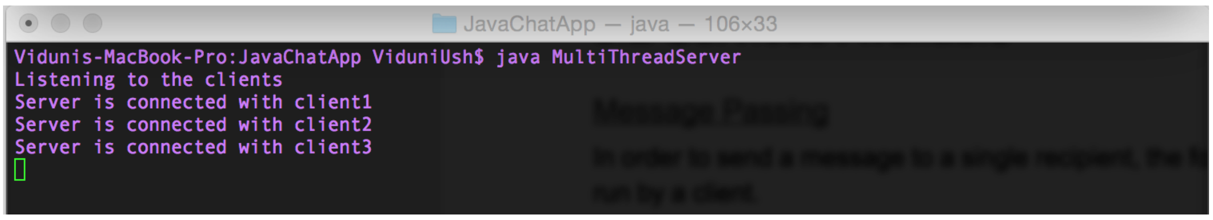
Figure 1 - Running the server



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 1
```

Figure 2 - Running the client (Client ID = 1)

In order to register multiple clients, separate terminal windows should be used for each client.



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadServer
Listening to the clients
Server is connected with client1
Server is connected with client2
Server is connected with client3
```

Figure 3 - When 3 clients are registered with the server using 3 separate terminals

Once the above steps are completed, the services provided can be consumed by service consumers and communication can be done through the middleware. Hence, the initial setup of the system is complete.

3. Workflow

3.1 Basic Architecture

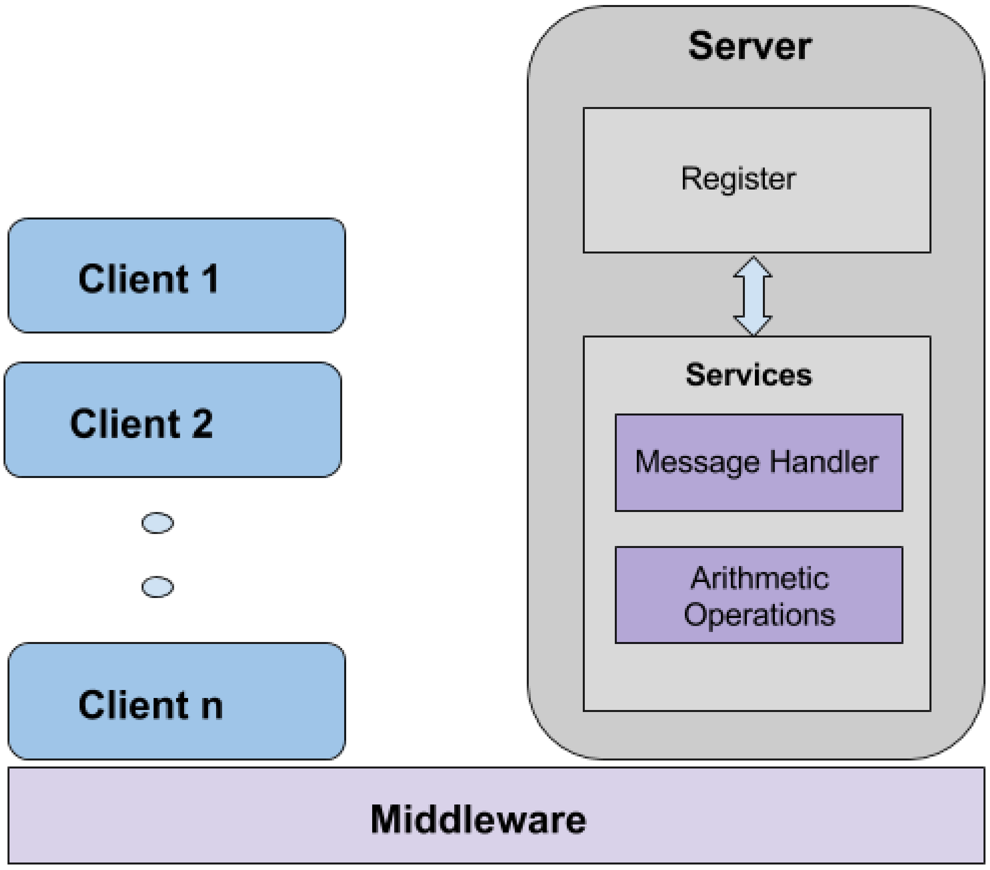


Figure 4 - Architecture

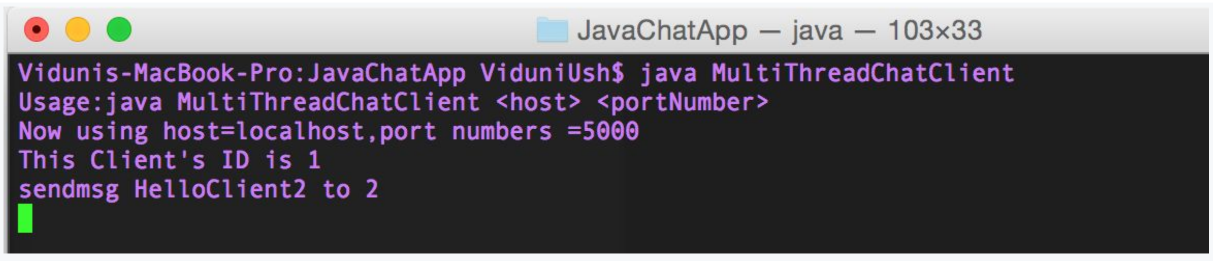
3.2 Services Available

Message Passing

In order to send a message to a single recipient, the following command should be run by a client.

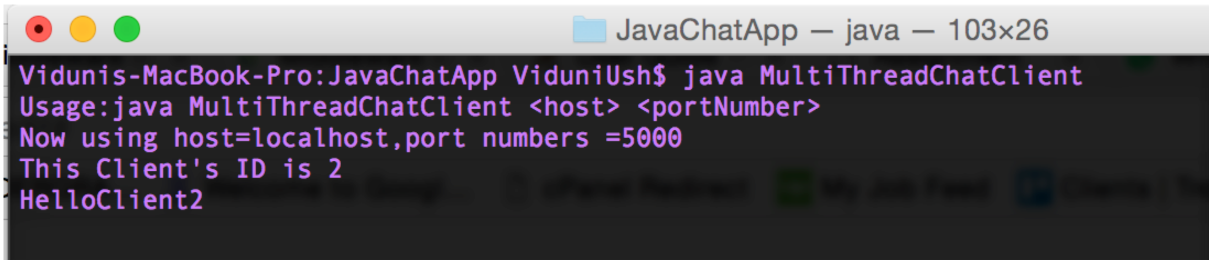
```
sendmsg <message> to <recipientID>
```

Eg: sendmsg HelloClient1 to 1



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 1
sendmsg HelloClient2 to 2
```

Figure 5 - Client 1 sending a message to client 2

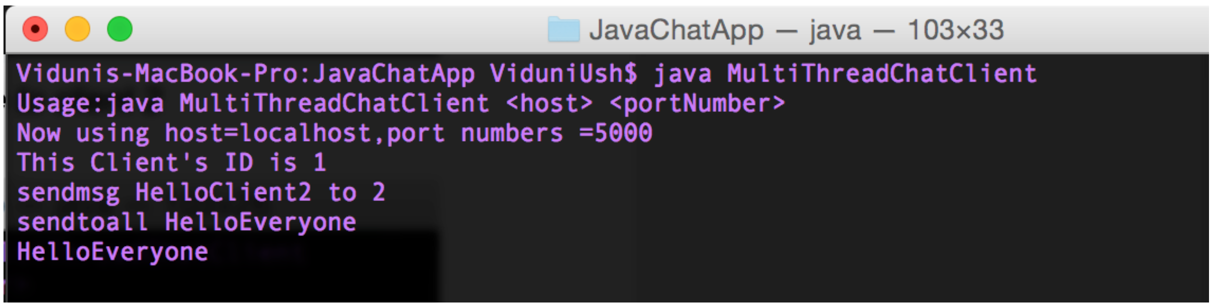


```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 2
HelloClient2
```

Figure 6 - Client 2 received the message from client 1

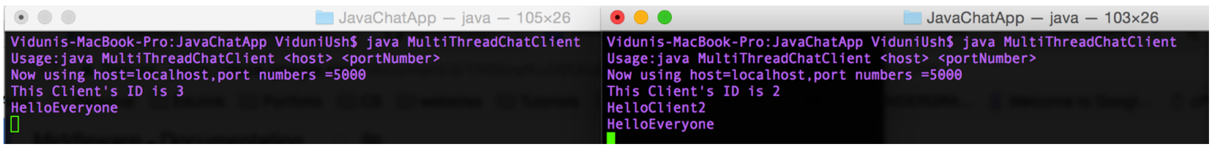
In order to send multiple requests (Messages to all other clients from one client), the following command needs to be used.

```
sendtoall HelloEveryone
```



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 1
sendmsg HelloClient2 to 2
sendtoall HelloEveryone
HelloEveryone
```

Figure 7 - Client 1 sends message to all other clients



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 3
HelloEveryone
```

```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 2
HelloClient2
HelloEveryone
```

Figure 6 - Both Client 2 and Client 3 (who is active currently) receives the message from client 1

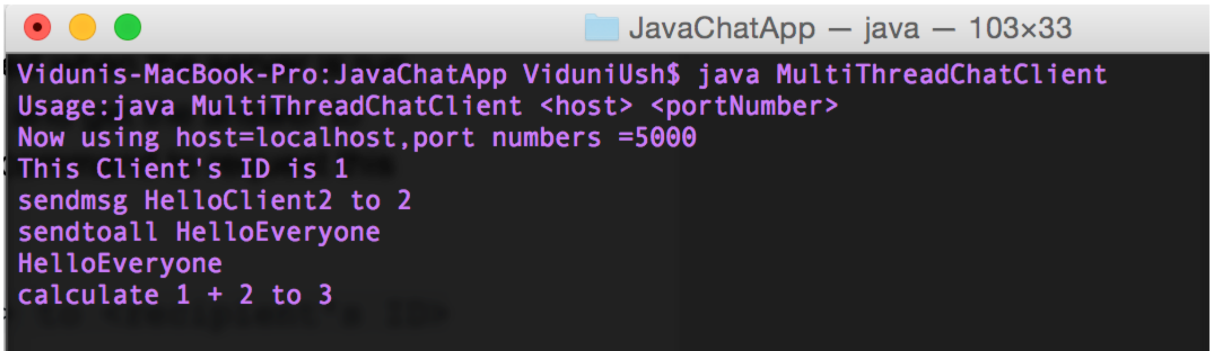
Arithmetic Operations

A service named, calculate is provided, which is registered when the server is run. Client can ask the server to perform a simple calculation and get the answer to himself/herself or send it to another specified client.The command to request this service is as follows.

```
calculate <operand1> <operator> <operand1> to <recipient's ID>
```

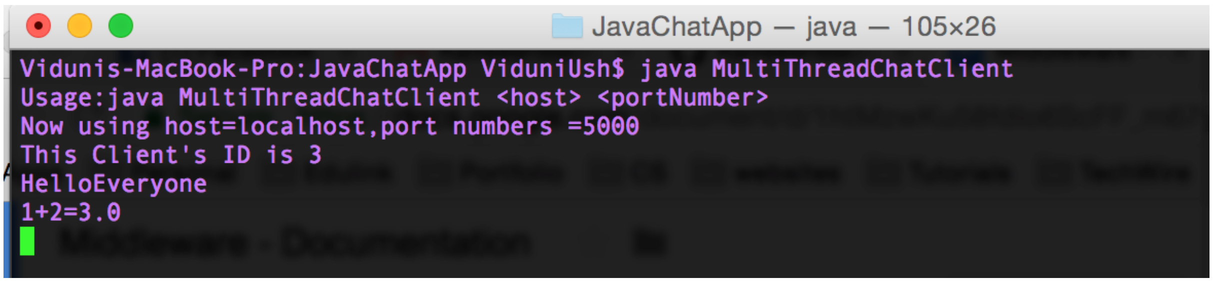
Eg: calculate 1 + 2 to 2

In here, the answer is calculated by the server and is sent to client 2. If a client wants to get an answer for a calculation, recipient's ID should be his/her own ID.



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 1
sendmsg HelloClient2 to 2
sendtoall HelloEveryone
HelloEveryone
calculate 1 + 2 to 3
```

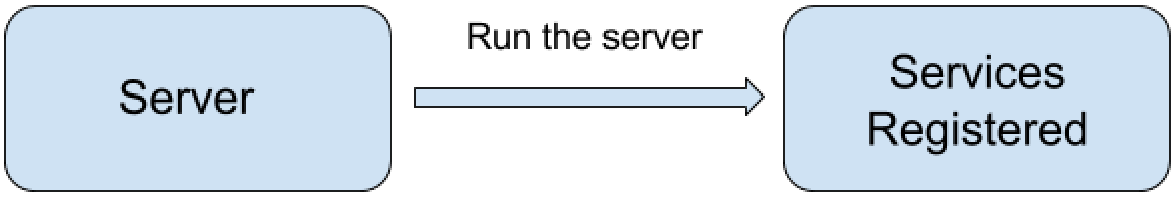
Figure 7-Client 1 requests an addition operation from the server to be sent to client 3



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 3
HelloEveryone
1+2=3.0
```

Figure 8 - Client 3 receives the answer of the addition operation

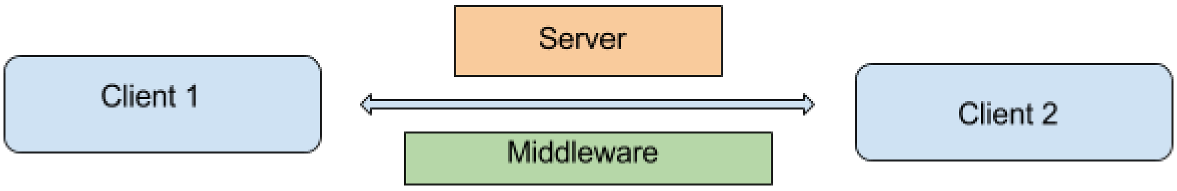
3.3 Registering a Service



3.4 Lookup a Service

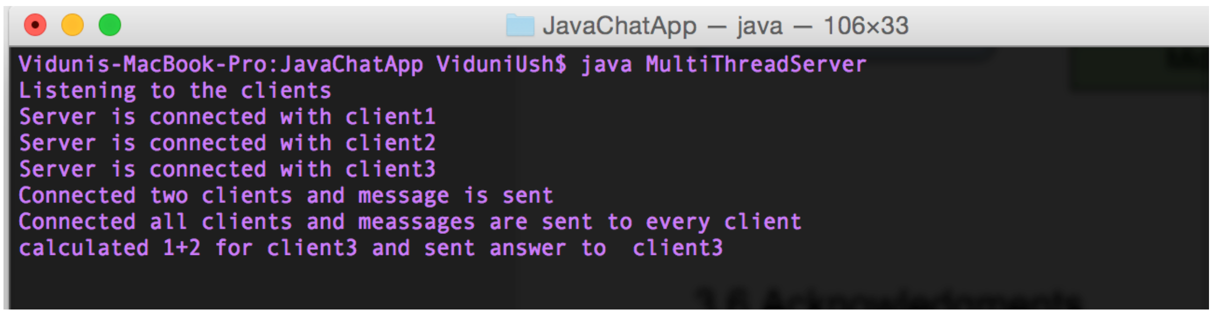
Pre-registered services have a specific message format in order to clients to request those services. The message format should be known by a client in order to request the particular service. An incorrect message format would result in an error.

3.5 Communication



3.6 Acknowledgments

When a client passes a message to another client via the middleware or when a client requests a service which is executed on the server, the server terminal provides acknowledgments in order to indicate the success or failure of different operations.

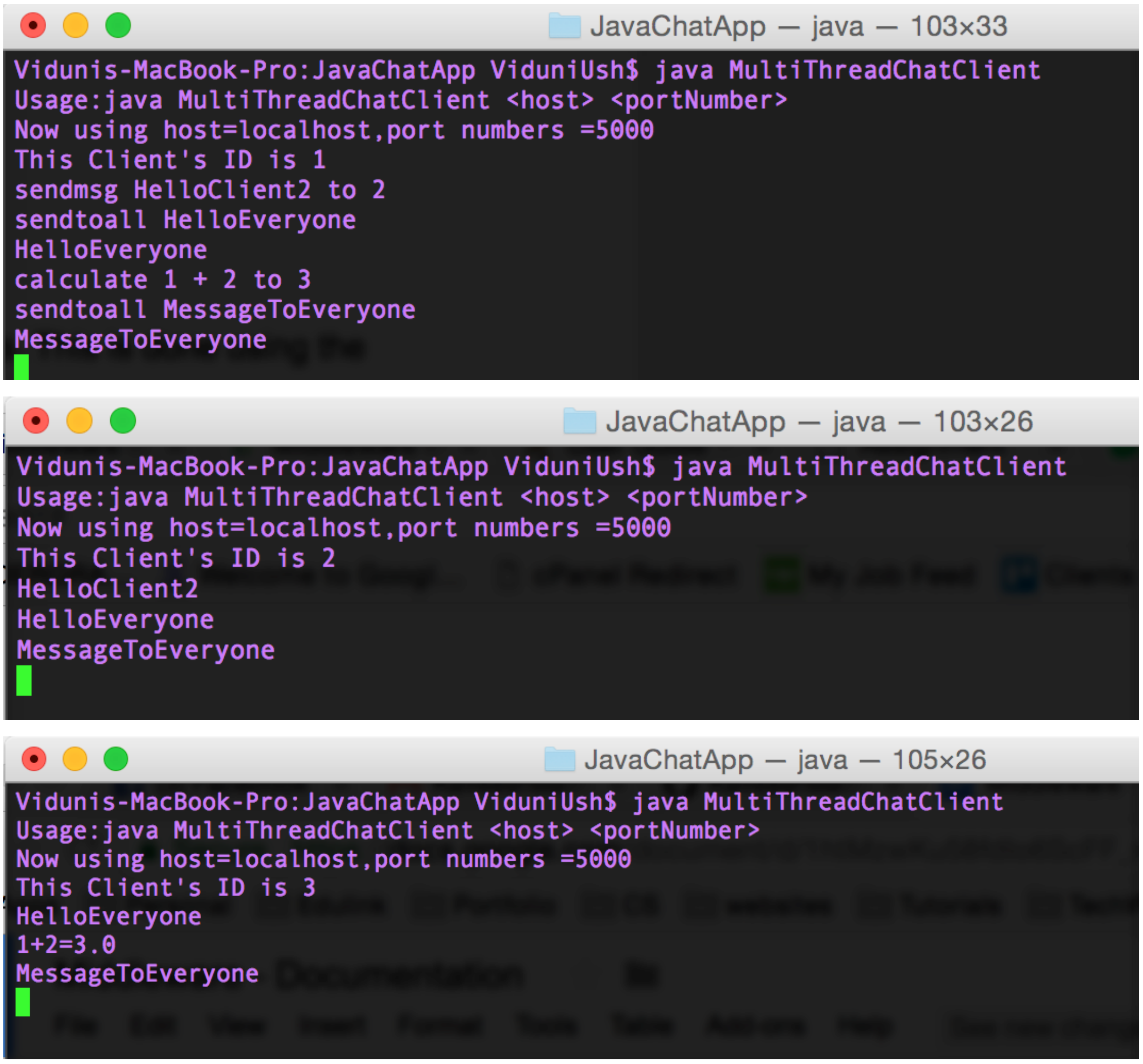


```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadServer
Listening to the clients
Server is connected with client1
Server is connected with client2
Server is connected with client3
Connected two clients and message is sent
Connected all clients and meassages are sent to every client
calculated 1+2 for client3 and sent answer to  client3
```

Figure 9 - Acknowledgments

3.7 Multiple Requests

A single client can send multiple requests to many clients. This is done using the `sendtoall` command.


JavaChatApp — java — 103×33
JavaChatApp — java — 103×26
JavaChatApp — java — 105×26

```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 1
sendmsg HelloClient2 to 2
sendtoall HelloEveryone
HelloEveryone
calculate 1 + 2 to 3
sendtoall MessageToEveryone
MessageToEveryone
```

```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 2
HelloClient2
HelloEveryone
MessageToEveryone
```

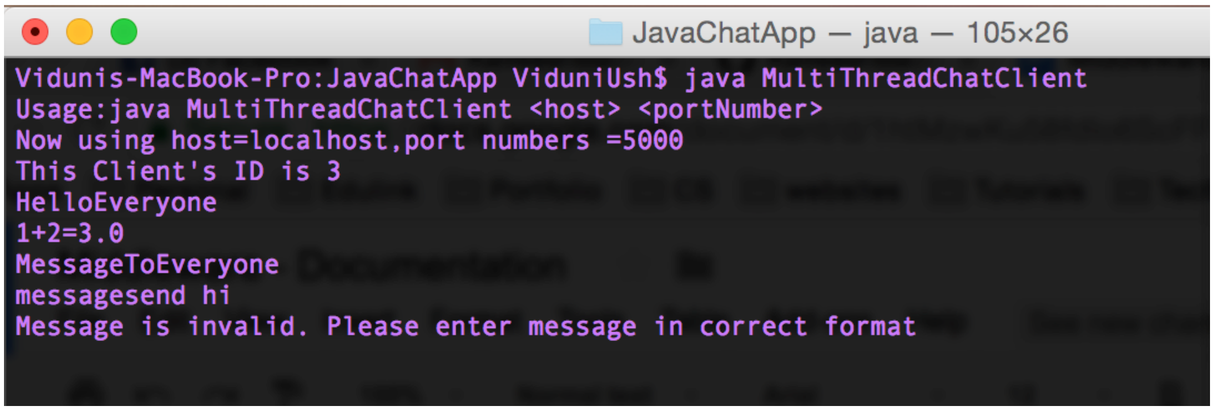
```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 3
HelloEveryone
1+2=3.0
MessageToEveryone
```

Figure 10 - Handling Multiple Requests

3.8 Error Handling

Incorrect Message Formats

When the correct message format is not followed by a client, an error is output in the client's terminal.

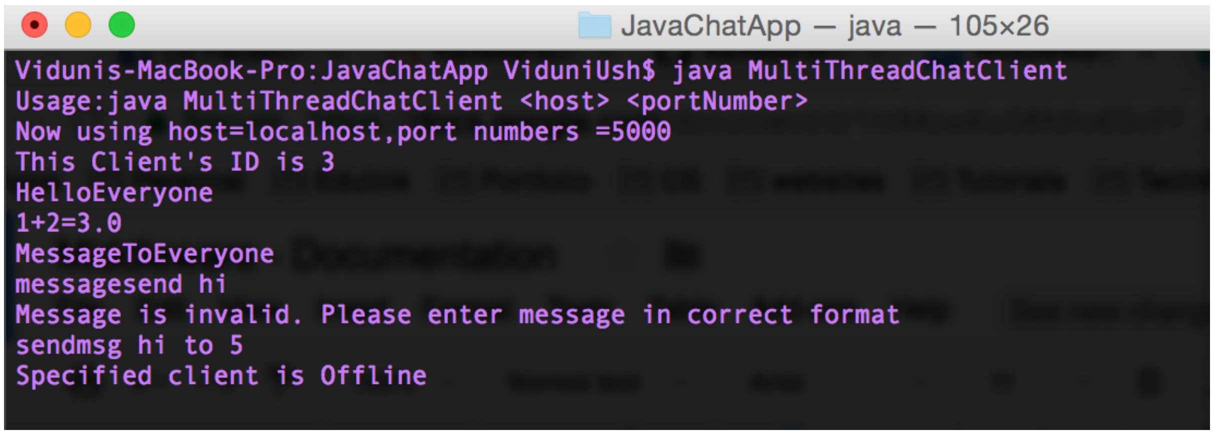


```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 3
HelloEveryone
1+2=3.0
MessageToEveryone
messagesend hi
Message is invalid. Please enter message in correct format
```

Figure 11 - Error when incorrect message format

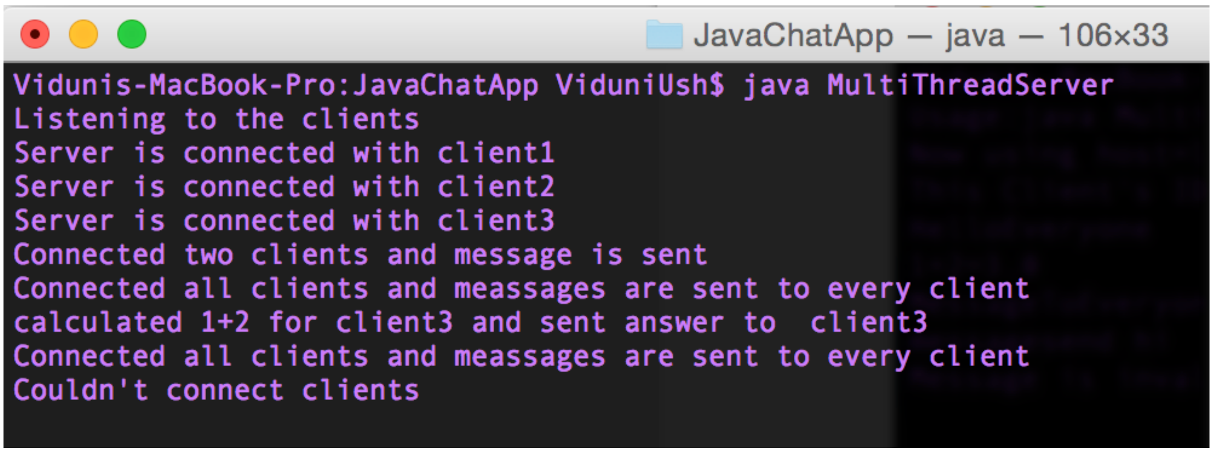
Client offline

When the requested client by another client is offline, it is notified to the client who requested communication with the offline client.



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadChatClient
Usage:java MultiThreadChatClient <host> <portNumber>
Now using host=localhost,port numbers =5000
This Client's ID is 3
HelloEveryone
1+2=3.0
MessageToEveryone
messagesend hi
Message is invalid. Please enter message in correct format
sendmsg hi to 5
Specified client is Offline
```

Figure 12 - Error on client's terminal when the requested client is offline (Specified client is offline)



```
Vidunis-MacBook-Pro:JavaChatApp ViduniUsh$ java MultiThreadServer
Listening to the clients
Server is connected with client1
Server is connected with client2
Server is connected with client3
Connected two clients and message is sent
Connected all clients and meassages are sent to every client
calculated 1+2 for client3 and sent answer to  client3
Connected all clients and meassages are sent to every client
Couldn't connect clients
```

Figure 13 - Error on server's terminal when the requested client is offline (Couldn't connect clients)