COMP 6231 - Distributed System Design

Assignment 3

(Option 1)

By Vasu Ratanpara (Student ID: 40135264)

◆ Distributed System Architecture
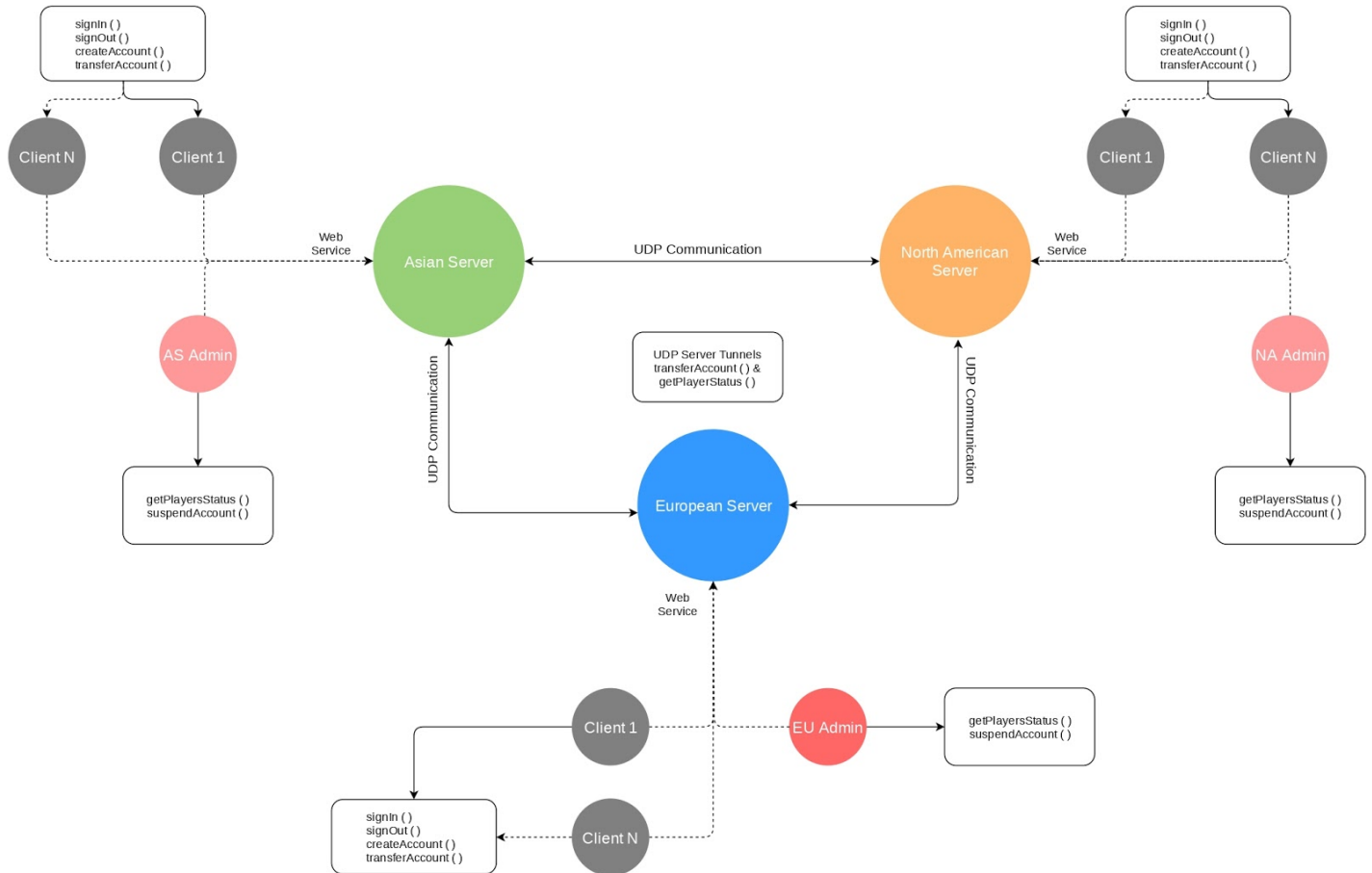

signIn ( )
signOut ( )
createAccount ( )
transferAccount ( )

Client N   Client 1

Web
Service

Asian Server   UDP Communication   North American Server

AS Admin

UDP Communication

UDP Server Tunnels
transferAccount ( ) &
getPlayerStatus ( )

UDP Communication

NA Admin

getPlayersStatus ( )
suspendAccount ( )

European Server

getPlayersStatus ( )
suspendAccount ( )

Web
Service

Client 1   EU Admin   getPlayersStatus ( )
suspendAccount ( )

signIn ( )
signOut ( )
createAccount ( )
transferAccount ( )

Client N

Figure 1.1: Architecture of a Distributed System

As you can see in the above figure, I tried to present a pictorial representation of Assignment 3 architecture. Here, we have 3 servers namely, Asian server, European server, and North American server. They all are connected with each other via UDP protocol. The whole assignment is based upon web services. Which is described on the next page.

For the Client program, the player can send a request to create an account, sign in, sign out, and transfer an account to another server. It will initially go to the appropriate server which will be decided by the IP address. Moreover, when a player will request a transfer of an account to different servers during that time the player's server will communicate with the receiving server via UDP and check if the account with the given

name already exists or not. The transfer account operation is an atomic operation. I have described it in-depth below in the report.

In addition, when the player will send a request for any operation it will redirect to the appropriate server as per their IP address. In the case of Admin, When Admin wants to know the status of players in the whole system. Admin will send a request to anyone server and that server will talk with the other 2 servers and request for their own status and send back to the Admin by combining all status of 3 servers.

❖ **Design of Web Services**
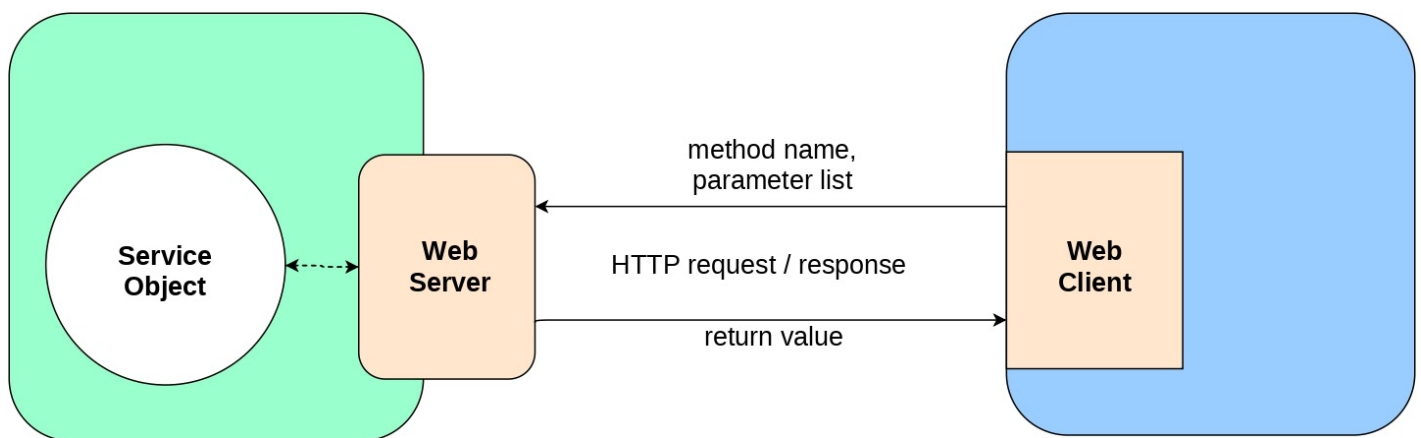
Figure 1.2: Design of web service (Ref. professor's PPT)

Web services run on HTTP protocol. To be more specific, "Web Services are software components described via WSDL which are capable of being accessed via standard network protocols such as SOAP/REST over HTTP/HTTPS."

❖ WSDL (Web Services Description Language):

A language based on XML notation to describe web services. WSDL is used to describe the location of the web service as well as the operations (methods, parameters and return values) of the service.

❖ SOAP (Simple Object Access Protocol):

A protocol originally defined by Microsoft, then standardized by the W3C. Uses XML notation to define the platform-independent inter-application information exchange mechanisms between consumers and web service providers.

Today, SOAP is the primary standard. SOAP provides rules for encoding the request and its arguments.

Similarly, the architecture doesn't assume that all access will employ HTTP over TCP. In fact, .NET uses Web Services "internally" even on a single machine. But in that case, communication is over COM.

WSDL documents are used to drive object assembly, code generation, and other development tools.

The major advantage of web services is they are platform independent as well as programming language independent. But the biggest disadvantage with web services are they are not as secure as CORBA and RMI because they run on HTTP and HTTPS.
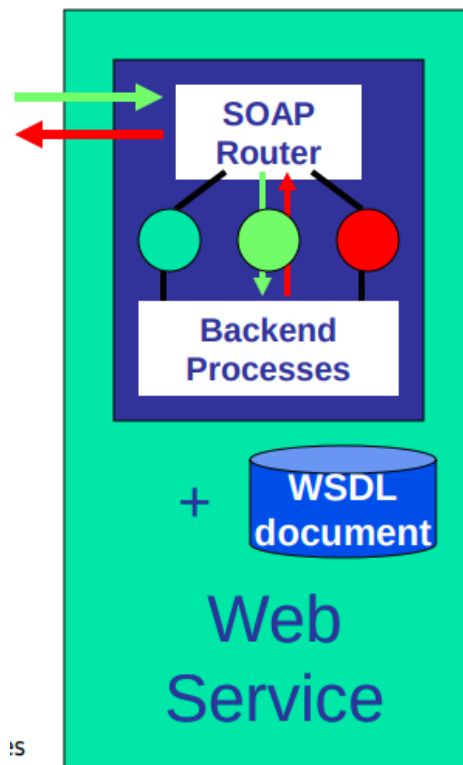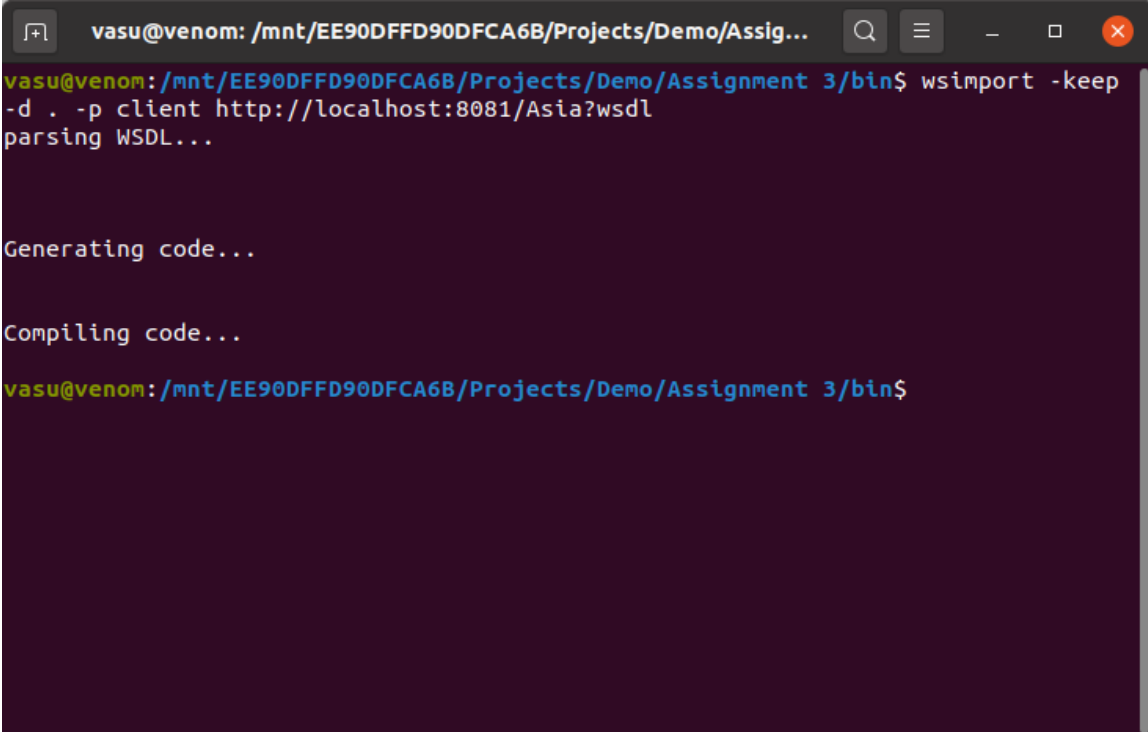
Figure 1.3: Design of web service in Java (Ref. professor's PPT)

◆ WSGEN and WSIMPORT commands

To generate the files for web service clients we can use wsimport command but we need to start our all servers so the web services can be published on endpoints.

wsimport -keep -d . -p client http://localhost:8081/Asia?wsdl

Figure 1.4: output of wsimport -keep -d . -p client http://localhost:8081/Asia?wsdl

wsimport -keep -d . -p client http://localhost:8082/Europe?wsdl

Figure 1.5: output of wsimport -keep -d . -p client http://localhost:8082/Europe?wsdl

wsimport -keep -d . -p client http://localhost:8083/NorthAmerica?wsdl

Figure 1.6: output of wsimport -keep -d . -p client http://localhost:8083/NorthAmerica?wsdl

wsgen -verbose -cp . server.AsianServerImpl -wsdl

Figure 1.7: output of wsgen -verbose -cp . server.AsianServerImpl -wsdl

wsgen -verbose -cp . server.EuropeanServerImpl -wsdl

Figure 1.8: output of wsgen -verbose -cp . server.EuropeanServerImpl -wsdl

wsgen -verbose -cp . server.NorthAmericanServerImpl -wsdl

Figure 1.9: output of wsgen -verbose -cp . server.NorthAmericanServerImpl -wsdl
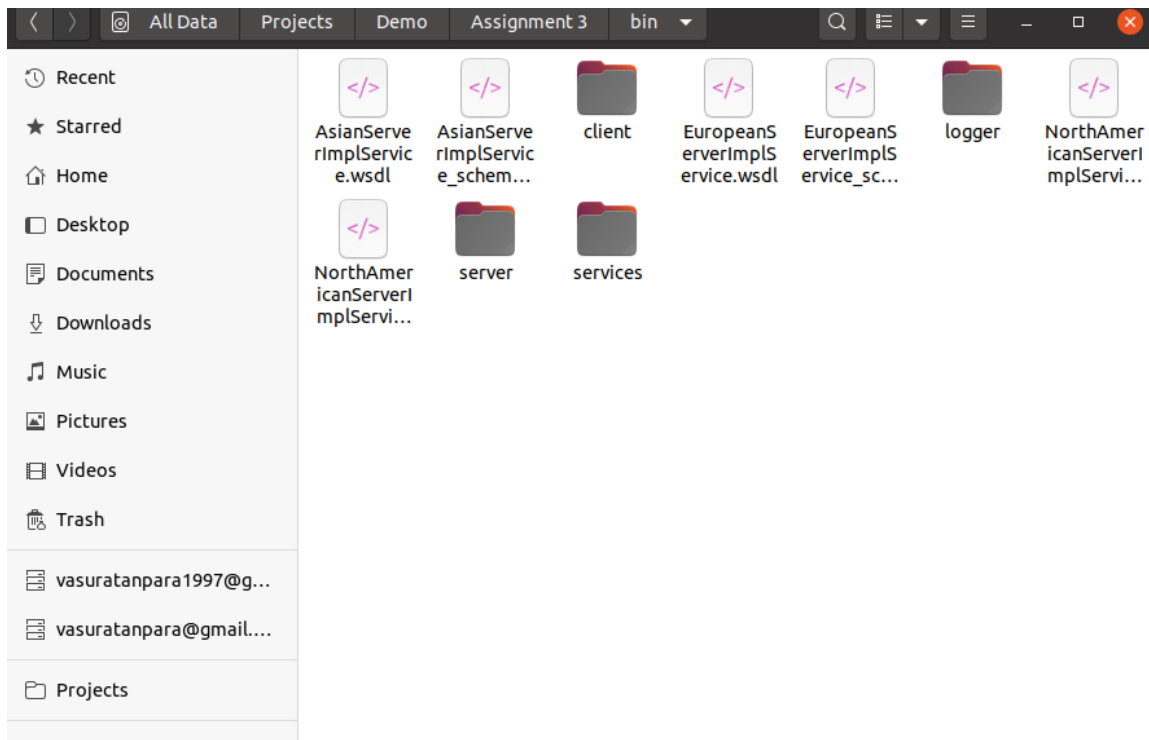

Figure 2.0: auto generated WSDL files

These commands will create files for web server clients which can be used by PlayerClient.java & AdministratorClient.java to connect with the servers.

All files which are created by these commands are stored in the 'services' package.

NOTE : All 3 servers have the same methods so I merged them in one package so, there I can reduce the redundant files in the same project.

◆ Distributed System Techniques & Data Structures

```
ConcurrentHashMap<String, ArrayList<HashMap<String, String>>> players
    = new ConcurrentHashMap<String, ArrayList<HashMap<String, String>>>();
```

To store the players' information on the servers, I have used Java's HashMap. Which will store the data in memory somewhat similar as described below?
{

"A": [<Player 1's HashMap>, <Player 2's HashMap>, ..., <Player N's HashMap>],

"B": [<Player 1's HashMap>, <Player 2's HashMap>, ..., <Player N's HashMap>],

"C": [<Player 1's HashMap>, <Player 2's HashMap>, ..., <Player N's HashMap>],

.

.

.

"Z": [<Player 1's HashMap>, <Player 2's HashMap>, ..., <Player N's HashMap>],

}

Here, the Keys are the First upper letter of the player's username and it would store an Array list of players' HashMap as a value whose first letter of the username is the same as Key. The structure of players' HashMap is as shown below.
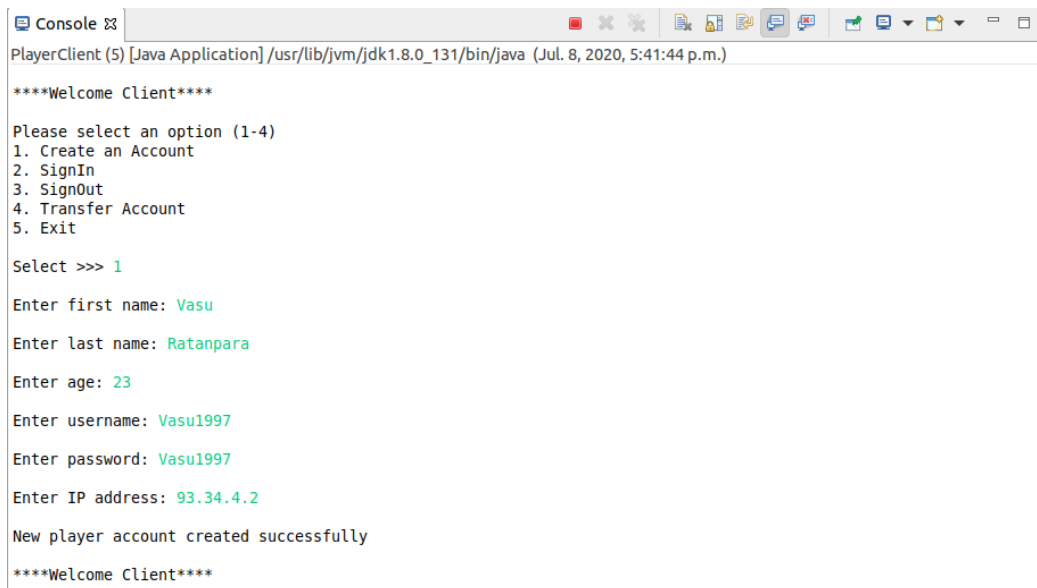
```
{
    "username": "Vasu123",
    "password": "Vasu1997",
    "firstname": "Vasu",
    "lastname": "Ratanpara",
    "age": 23,
    "ipaddress": "182.45.12.3",
    "status": "offline",
}
```

This player will be stored in the Array list of Key "V" because his username is starting with "V".

Moreover, I used ConcurrentHashMap<K,V> which is part of Java's in-built package java.util.concurrent. It has full concurrency support and it is a thread-safe data structure so that only one thread can access it at a time.
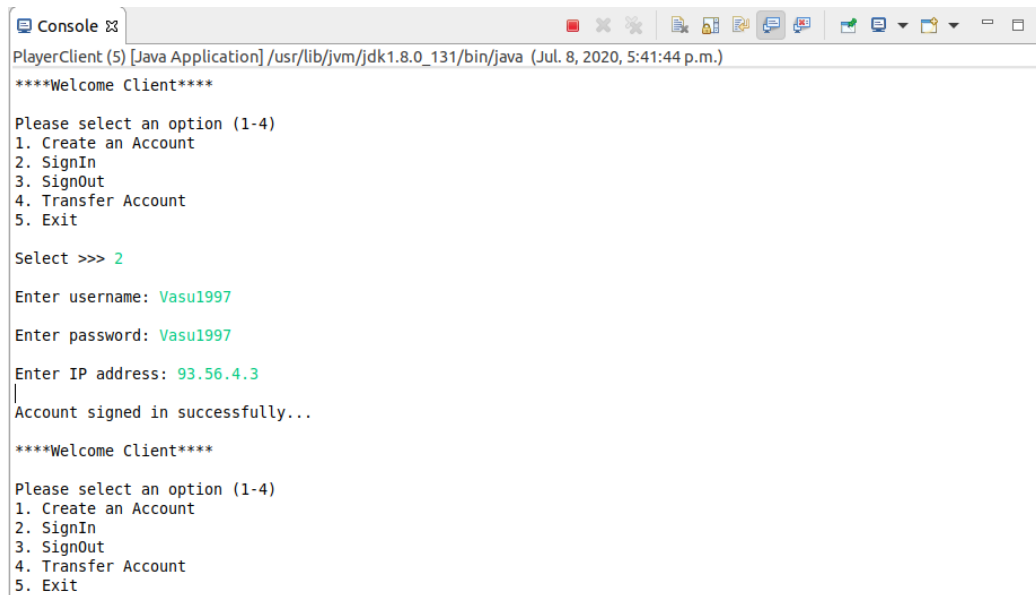
◆ Test Case Scenarios & Explanation

1. Create an account

Figure 2.1: Create an account

Here as you can see a player can register a new account with the PlayerClient program. All fields are mandatory and the username and password field has length validations which will allow a minimum length of the field as 6 characters. The user account will be stored on the server which is relevant to the given IP address. If the user is already

present with the same username then it will give an error that will not create an account.
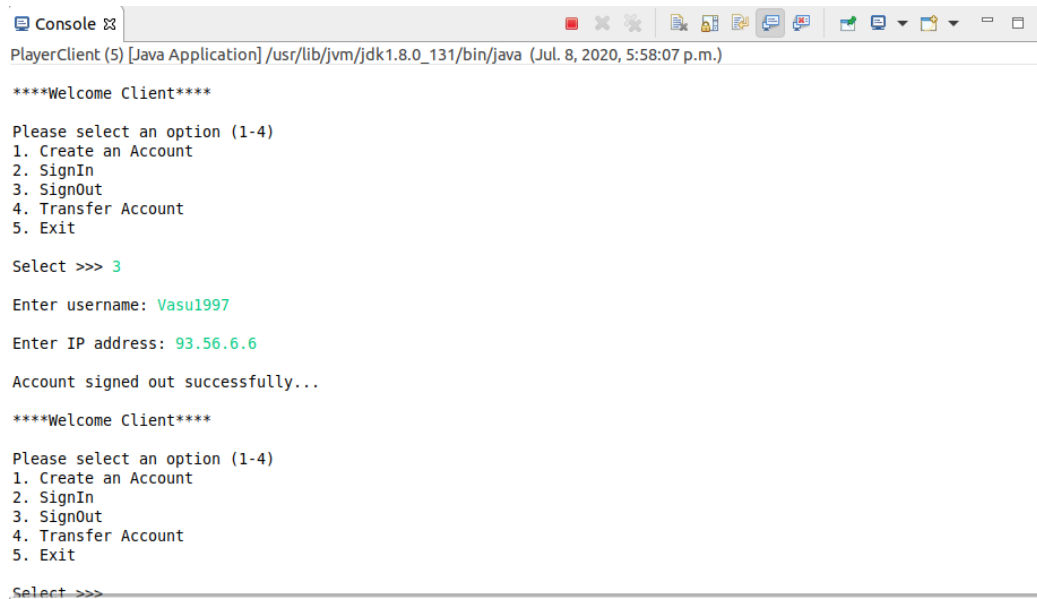
2. Sign in an account

Figure 2.2: Sign in an account

To sign in an account the player needs to provide a username, a password, and the IP address of his/her associated server. For example, the request will be redirected to the European server because the IP address is of Europe I.e. starting with 132.xxx.xxx.xxx. If the password is wrong or the user with the given username doesn't exist, then the server will discard the request and return an appropriate error. If the username and password are valid then the status of the user will be online in the server.

3. Sign out an account

Figure 2.3: Sign out an account

Sign out operation is straightforward. The player just needs to provide a username and the IP address of his/her location. If the user with given username exists and online then it will be set to an offline and sign out message will be sent to the player.
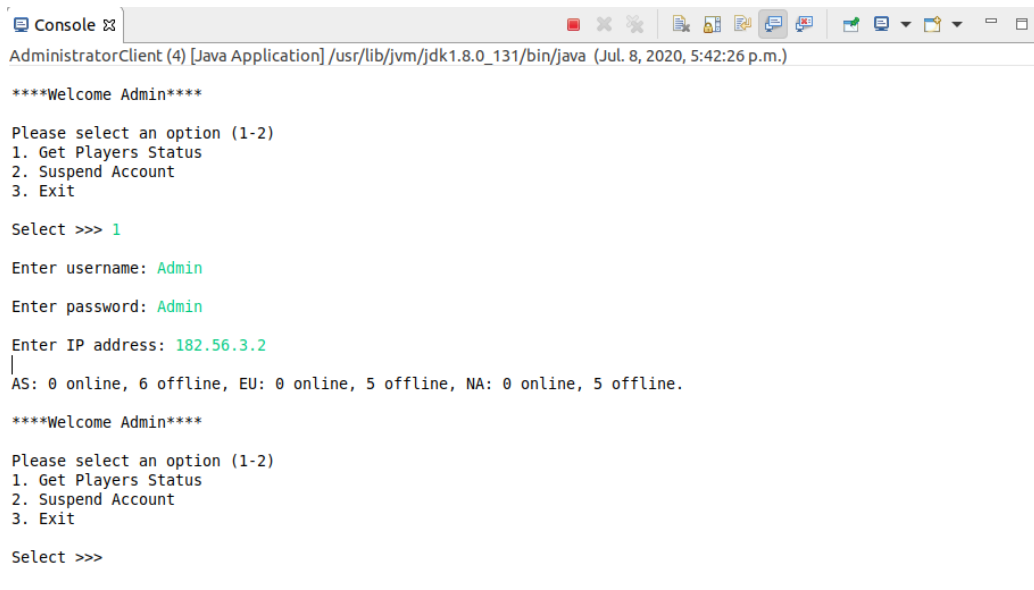
4. Transfer an account

Figure 2.4: Transfer an account

The transfer account will take 4 parameters namely, username, password, IP address (Old IP address), and New IP address. The account will be transferred from an old server to a new server (both will be detected by each IP separately). The whole process is explained in-depth under an atomicity section.

5. Get the players' status

Figure 2.5: Get players' status

This operation will display the status of all players in the distributed system. It will ask for an Admin's username and password and the IP address on any of one server. The request will go to that server and that server will send UDP requests to the other two

servers to know their players' status. Once it will get responses it will merge the response and send it to an AdminClient.

6. Suspend an account
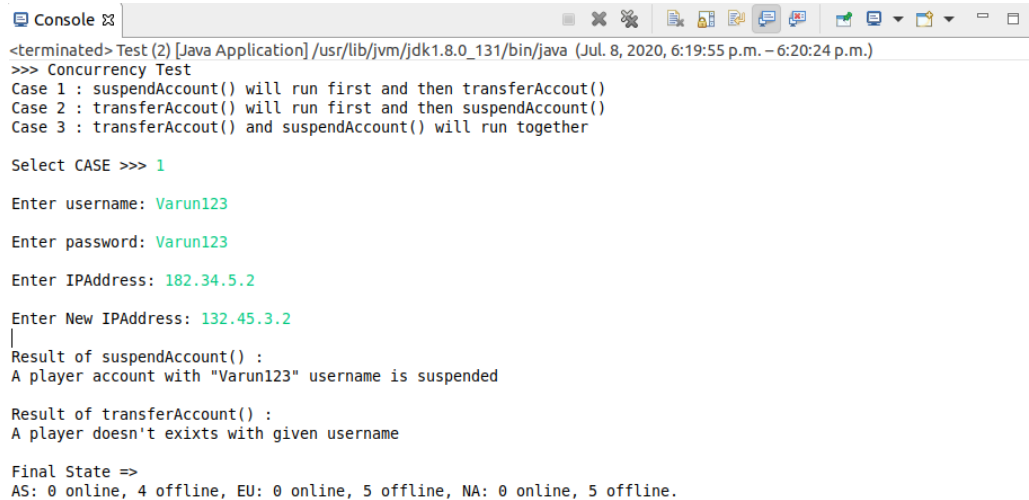
Figure 2.6.: Suspend an account

The suspended account will also take an Admin's username and password. Moreover, it will also take an IP address of a server and the username which needs to be suspended. The suspended account will delete the user with the given username and send a response back to AdminClient. If the user doesn't present the server with a given username it will send an error message.

7. Concurrency test (Case 1)

Here, I have created a new Java file called Test.java in the Client package. Which will test concurrency in scenario 7,8 and 9th. In case 1, the suspendAccount() will execute first and after 1 second the transferAccount() will be called. To simulate the real-life scenario, I have used the Thread.sleep() function. The output would be as shown above the account will be suspended and when another thread will try to transfer an account it will give an error.

Figure 2.7: Concurrency Test (Case 1)

8. Concurrency test (Case 2)

Figure 2.8: Concurrency Test (Case 2)

This case is the vice versa of case 1. Here the transferAccount() will run first and after 1 second delay suspendAccount() will run. So, the user account would be transferred to another server before it gets suspended.

9. Concurrency test (Case 3)

This is a very special case for concurrency testing. I removed synchronized keywords from transferAccount() and suspendAccount() methods from all 3 servers to see what will happen if the server gets two requests at the same time and how it will be handled by the server.

Figure 2.9: Concurrency Test (Case 3)

As you can see in the screenshot, both threads executed at the same time so when the account was transferring from one server to another server meanwhile that account was suspended by an Admin. So, when the response from server 2 will come and server 1 will try to remove that user from the database. But that user was suspended by the Admin so deletion operation will fail, and it will start the rollback process. In this process, it will send a request to server 2 to remove that user's account because it was suspended by an Admin when it was transferring, and the rollback will be successful after that.

Note: To test this case you must need to remove a synchronized keyword from transferAccount() and suspendAccount() methods from all 3 servers

10. Other Validations (IP, Age, Username and Password)

- IP validation

```
****Welcome Client****

Please select an option (1-4)
1. Create an Account
2. SignIn
3. SignOut
4. Transfer Account
5. Exit

Select >>> 2

Enter username: Kevin123

Enter password: Kevin123

Enter IP address: 147.34.2.1

1. 132.xxx.xxx.xxx : a North-American geo-location.
2. 93.xxx.xxx.xxx  : an European geo-location.
3. 182.xxx.xxx.xxx :  an Asian geo-location.

Invalid IP address
```

Figure 3.0: IP validations

- Age, Username and Password validations

```
Enter first name: Kim

Enter last name: John

Enter age: I don't know

Please input Integer only

Enter age: 25

Enter username: Kim25

A username must have at least 6 characters
Enter username:
KimJohn

Enter password: KimJ

A password must have at least six characters
Enter password:
KimJohn
```

Figure 3.1: Age, Username and Password validations

◆ important/Difficult Part in The Assignment

1. Atomicity

Atomicity in transferAccount( )
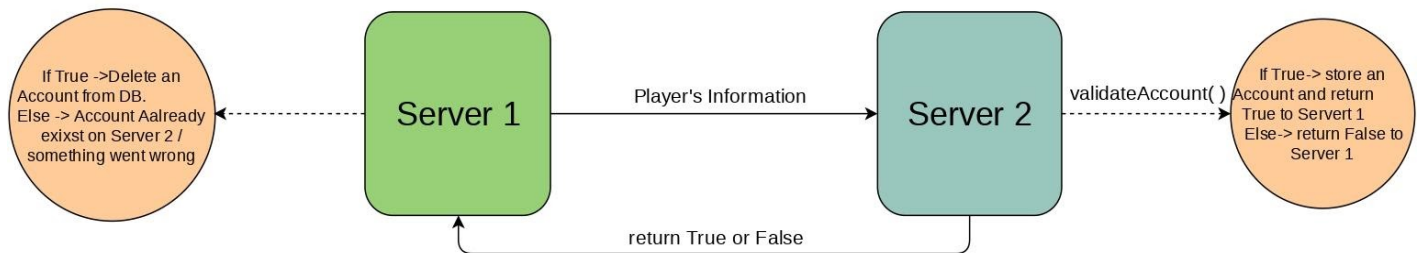
Figure 3.2: Atomicity in transferAccount()

The important part of the assignment was to achieve atomicity in transferAccount() operation. It can be done in multiple ways. In the above diagram, I tried to show a simple way to achieve atomicity.

When the server 1 ger account transfer request from a player it will send a new request to server 2 via UDP tunnel with all data of the user. When server 2 will receive a UDP request it will validate an account and check if any user is already present with the receiver user's username if the user with the same name is present then the server 2 will send "False" to server 1 and server 1 will send a message to the client that the user already exists with given username. Else, it will create a new account for that transferred user and return "True" to server 1.

If server 1 will receive a "True" from server 2 then it will delete that user's account from the database. For any reason If the deletion operation won't succeed then, the rollback process will start, and it will bring the system in its original state before the transfer operation was started.

2. ConcurrentHashMap Vs. HashTable in Java

To maximize the concurrency and make the data structure more thread-safe I decided to use ConcurrentHashMap which gives better performance. ConcurrentHashMap only locks the data structure while writing.

3. Creating Web Services files using WSDL and WSGEN commands

At first, it was a little bit hard for me to figure out how to use both commands but I tried to create web services using SOAP binding annotation. But later I figured out how to use **wsimport** and **wsgen** commands.

Eventually, I was able to clear out similar files which were generated by both commands for the 3 servers. Because all 3 servers are identical so the created files were identical as well.

◆ **Distributed System UML Design**
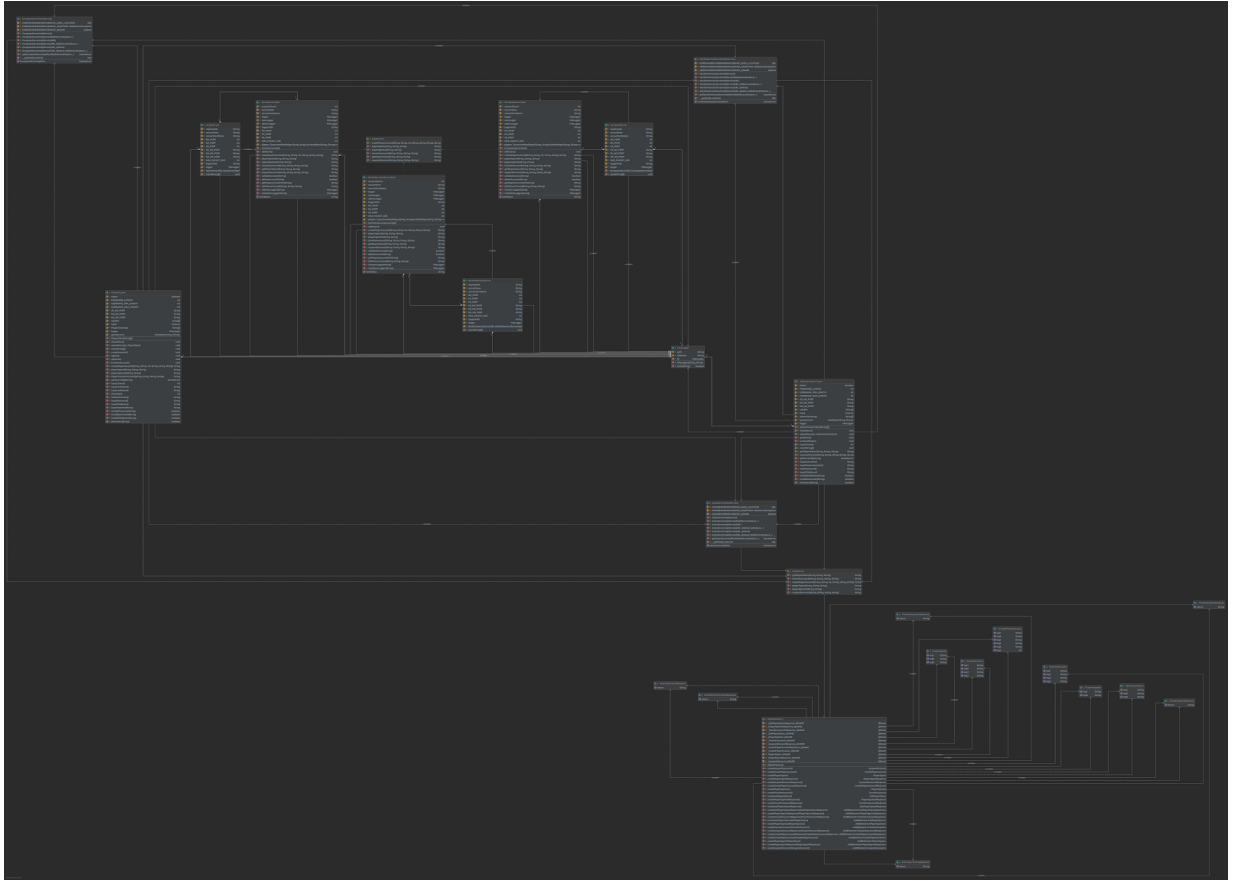
Figure 3.4: UML diagram

The figure represents the UML class diagrams. In figure all classes are covered, and it represents the different relation between all classes.

Note: To see the diagrams properly please goto 'uml' folder which is inside the Assignment 3 folder.

◆ **Project's Defaults**

● **Web Services Ports**

AS_WS_PORT = 8081, EU_WS_PORT = 8082, NA_WS_PORT = 8083

● **UDP Server Ports**

NA_PORT = 5001, EU_PORT = 5002, AS_PORT = 5003

Note: Please kill all services which are running on the above port before running the project.

◆ Asian Server's Default Users (182.x.x.x)

Sr. No	Username	password
1	Bruce123	Bruce123
2	Charles123	Charles123
3	Adak123	Adak123
4	Varun123	Varun123
5	Kevin123	Kevin123

◆ European Server's Default Users (93.x.x.x)

Sr. No	Username	password
1	Fabienne123	Fabienne123
2	Darcie123	Darcie123
3	Philipa123	Philipa123
4	Davinia123	Davinia123
5	Gyles123	Gyles123

❖ North American Server's Default Users (132.x.x.x)

Sr. No	Username	password
1	Siddall123	Siddall123
2	Morce123	Morce123
3	Seabrook123	Seabrook123
4	Upton123	Upton123
5	Garfield123	Garfield123

◆ How to run a project?

The project was created using Oracle's Java 8.

Unzip project submission. There will be 1 file and 1 folder namely "Report.pdf" and "Assignment 3" folder. Put the "Assignment 3" folder in your Eclipse workspace and open the project in your Eclipse.

There are 4 packages in the folder namely client, server, services and logger.
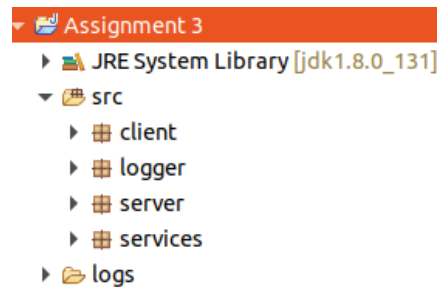
Figure 3.5: Assignment 3 packages

❖ <u>src folder</u>

1. Client package
   - AdministratorClient.java
   - PlayerClient.java
2. Services package (Auto-generated java files with **wsgen** command)
3. Server package
   - GameServer.java (Interface)
   - AsianServer.java (Asian Server)
   - AsianServerImpl.java
   - EuropeanServer.java (European Server)
   - EuropeanServerImpl.java
   - NorthAmericanServer.java (North American Server)
   - NorthAmericanServerImpl.java
4. Logger package
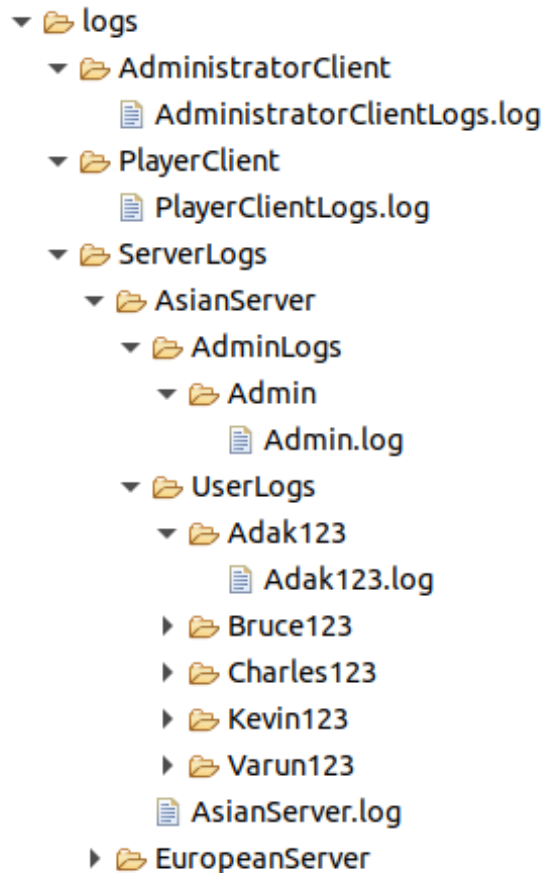   - FileLogger.java

◆ <u>logs</u> folder

Figure 3.6: Log Folder Structure

1. PlayerClient
   - PlayerClientLogs.log
2. AdministratorClient
   - AdministratorClientLogs.log
3. ServerLogs
   - NorthAmericanServer
     1. NorthAmericanServer.log
     2. UserLogs (It has all North American users' log files)
   - EuropeanServer
     1. EuropeanServer.log
     2. UserLogs (It has all European users' log files)
   - AsianServer
     1. AsianServer.log
     2. UserLogs (It has all Asian users' log files)

❖ <u>WSDL</u> folder

all WSDL files which are created by wsgen commands are located in the wsdl folder.
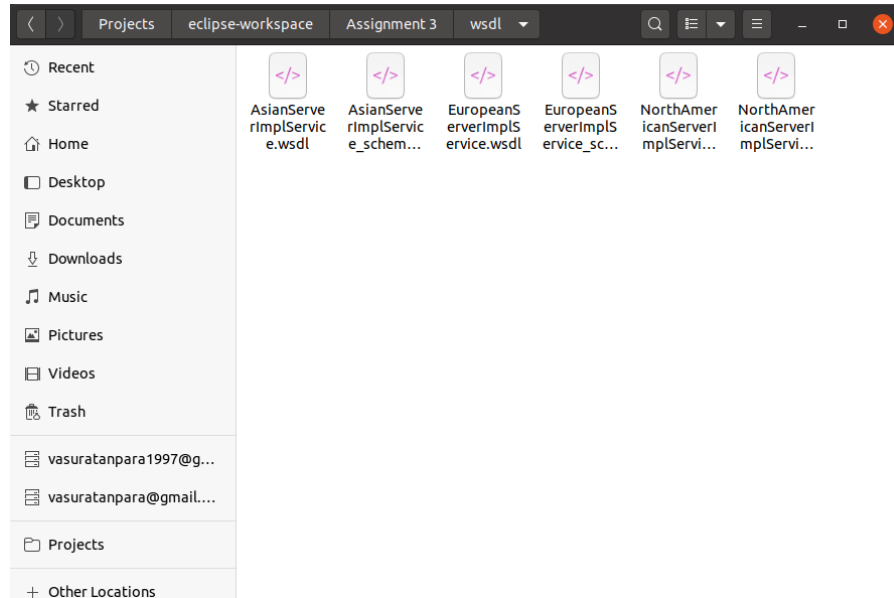

Figure 3.7: WSDL Folder

❖ <u>UML</u> folder

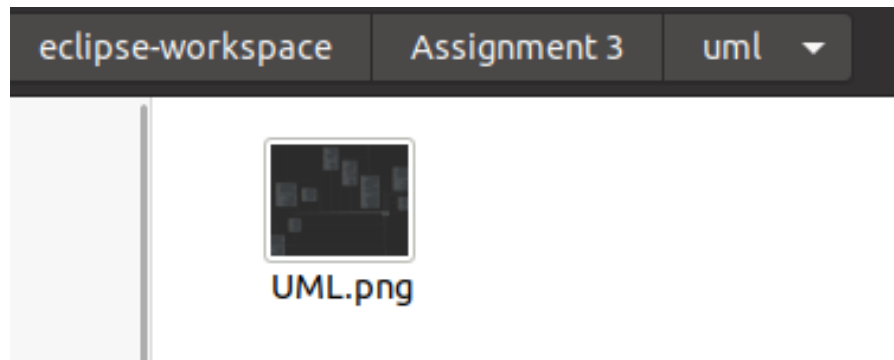UML diagram is stored inside the UML folder which is located in the Assignment 3 folder.

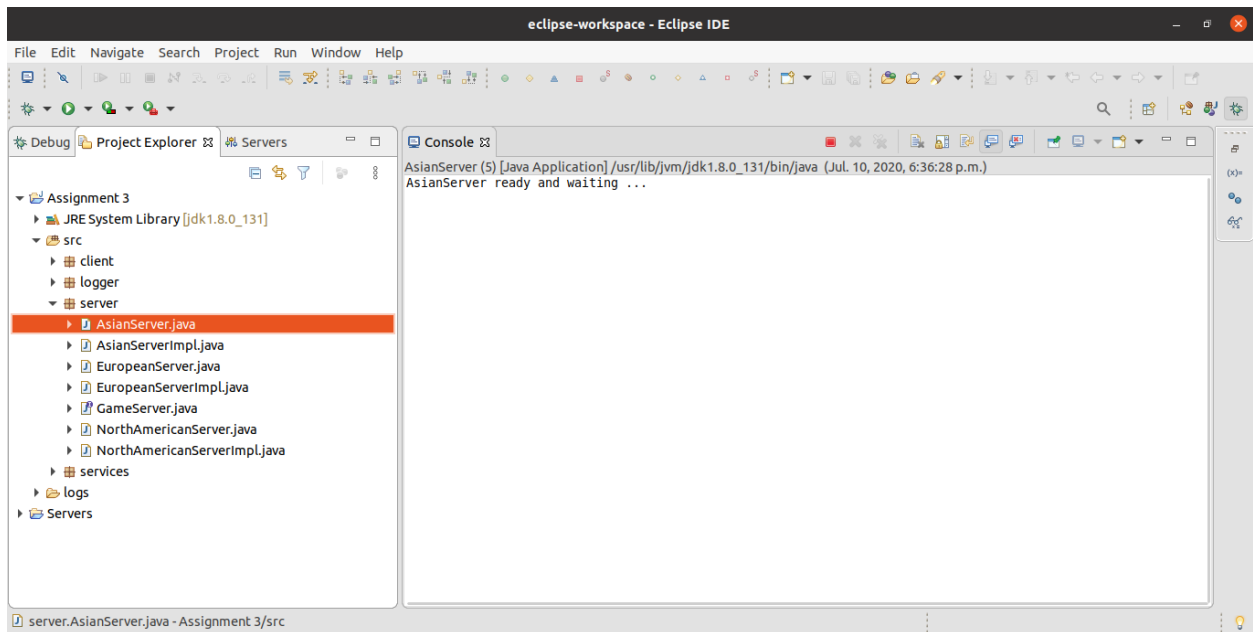Figure 3.8: UML diagrams

◆ Staring Asian Server

Figure 3.9: Starting the Asian server

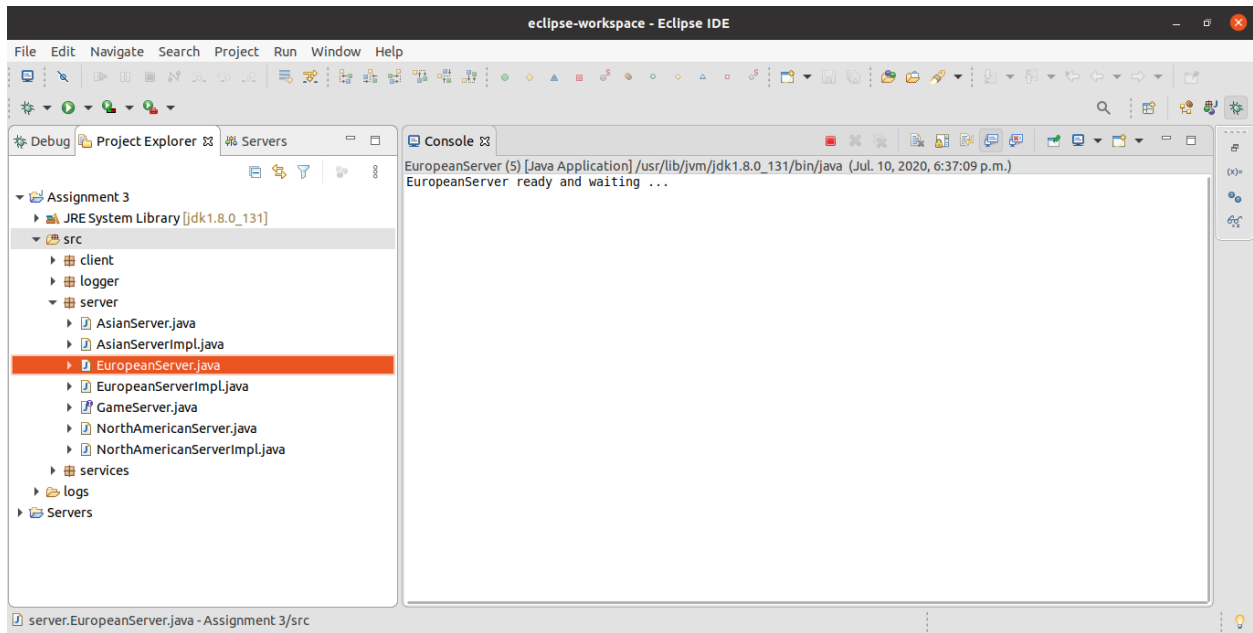## ◆ Starting European Server

Figure 4.0: Starting the European server
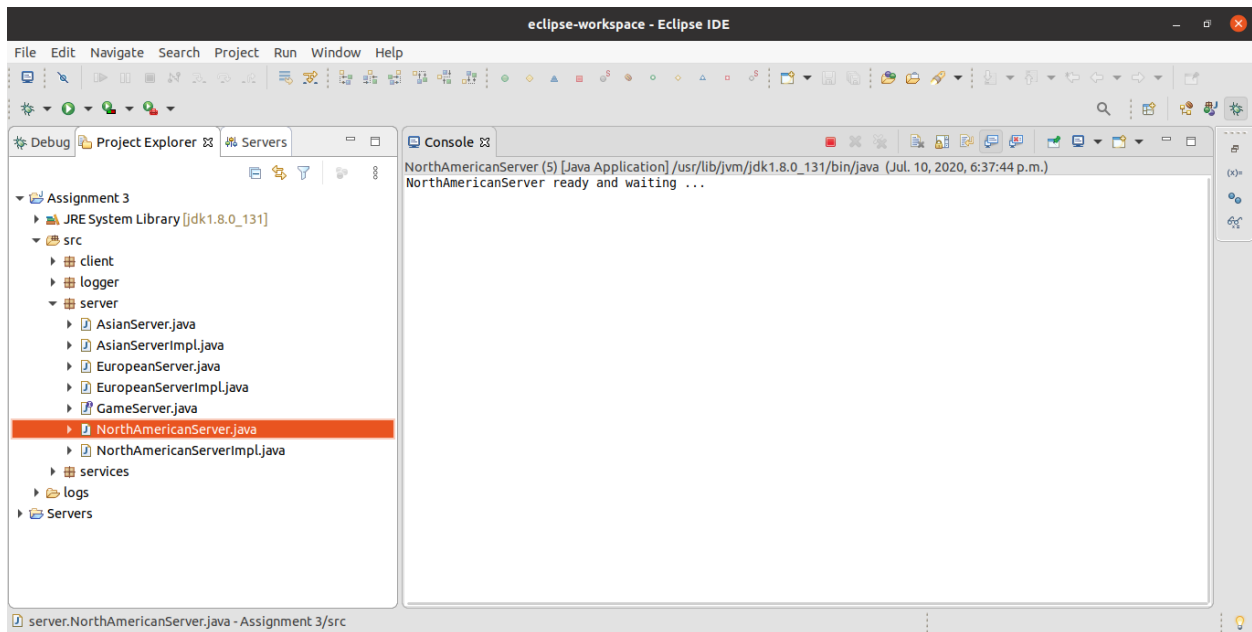
◆ Starting North American Server

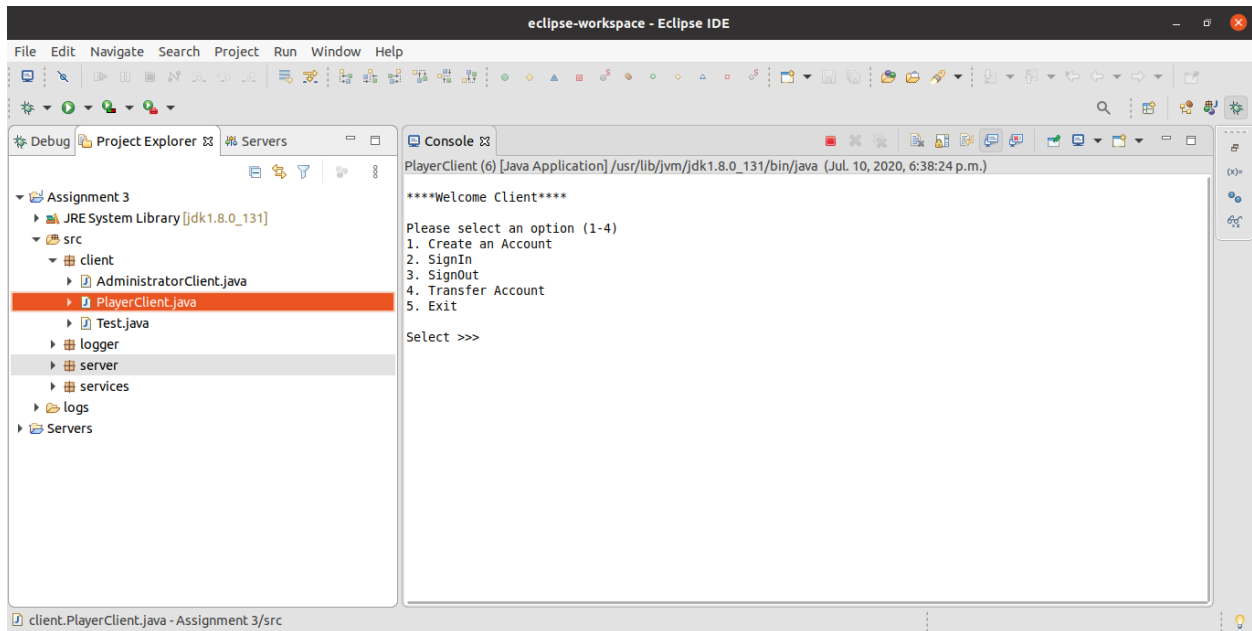Figure 4.1: Starting the North American server

◆ Starting PlayerClient

Figure 4.2: Starting the PlayerClient
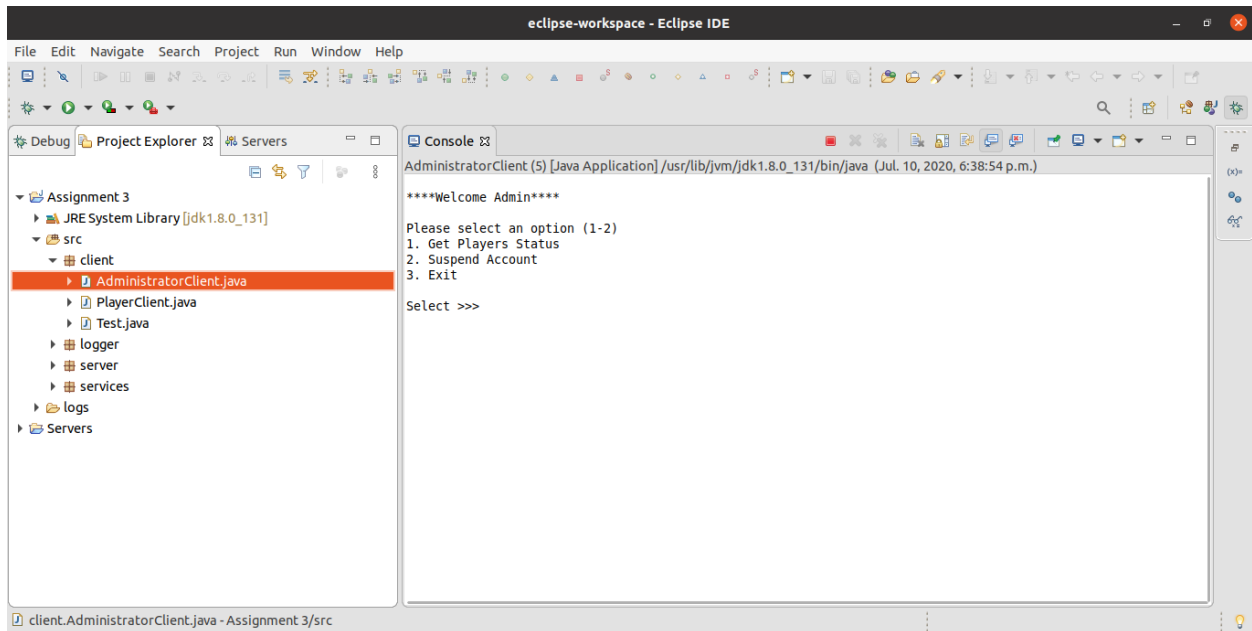
❖ Staring AdministratorClient

Figure 4.3: Starting the Admin Client

❖ Logs

Logs folder will be created in the eclipse project folder. Here, you can see "logs" folder with "src" and "bin" folder
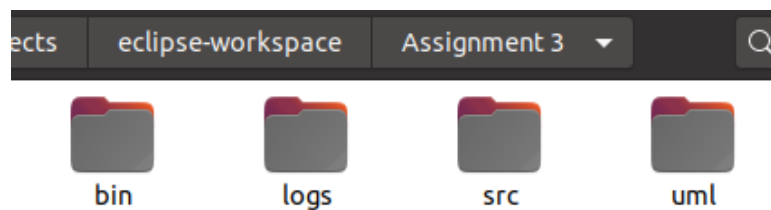

Figure 4.4: Assignment 2 folder structure

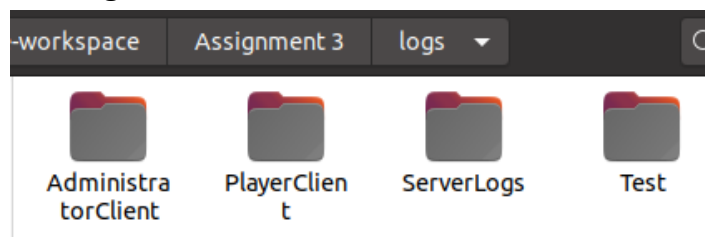There are 3 folders inside "logs" folder

Figure 4.5: Logs folder structure

Inside the "ServerLogs" folder there is a separate folder for each server that contains the logs for individual servers and their users.
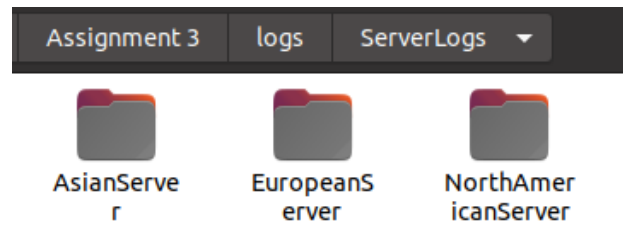

Figure 4.6: ServerLogs folder structure

Server's log file will be stored outside and Admin's log are stored in "AdminLogs" folder and users' logs will be stored in "UserLogs" folder
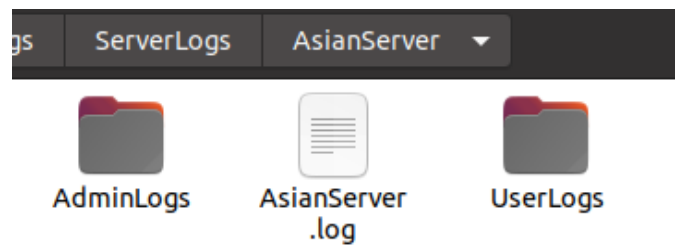

Figure 4.7: AsianServer folder structure

Here, as you can see there are 5 folders for 5 different users (by username) each folder contains individual log files for each user
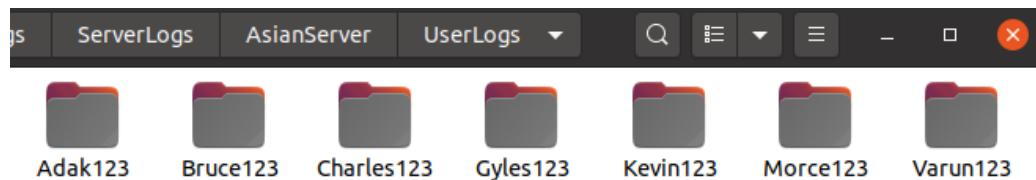

Figure 4.8: UserLogs folder structure

User log for a player with username "Varun123".
● Feel free to contact me if you do not understand anything or you have any questions. My email address is vasu.ratanpara@mail.concordia.ca

Thank You