

# COMP 6231 - Distributed System Design

## Assignment 2

By Vasu Ratanpara (Student ID: 40135264)

## ❖ Distributed System Architecture

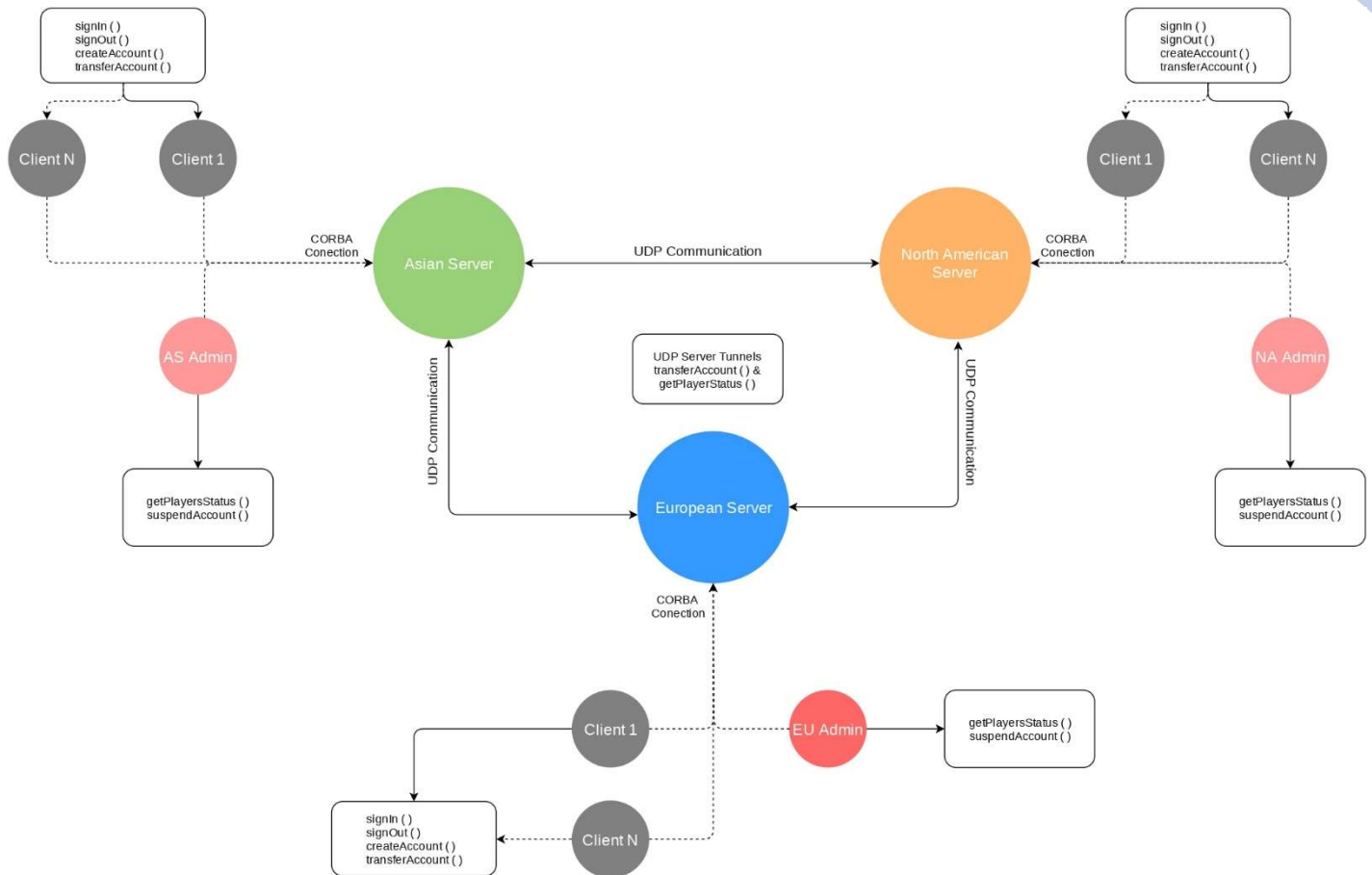


Figure 1.1: Architecture of Distributed System

As you can see in above figure, I tried to present pictorial representation of the Assignment 2 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 CORBA architecture. Which is described in next page.

For the Client program, player can send a request for create an account, sign in, sign out and transfer an account to another server. It will initially go to appropriate server which will be decided by the IP address. Moreover, when a player will request for a transfer an account to different server during that time the player's server will communicate with receiving server via UDP and check if the account with given name is already exist or not. The transfer account operation is an atomic operation. I have described it in-depth below in the report.

## Project Request Broker Architecture

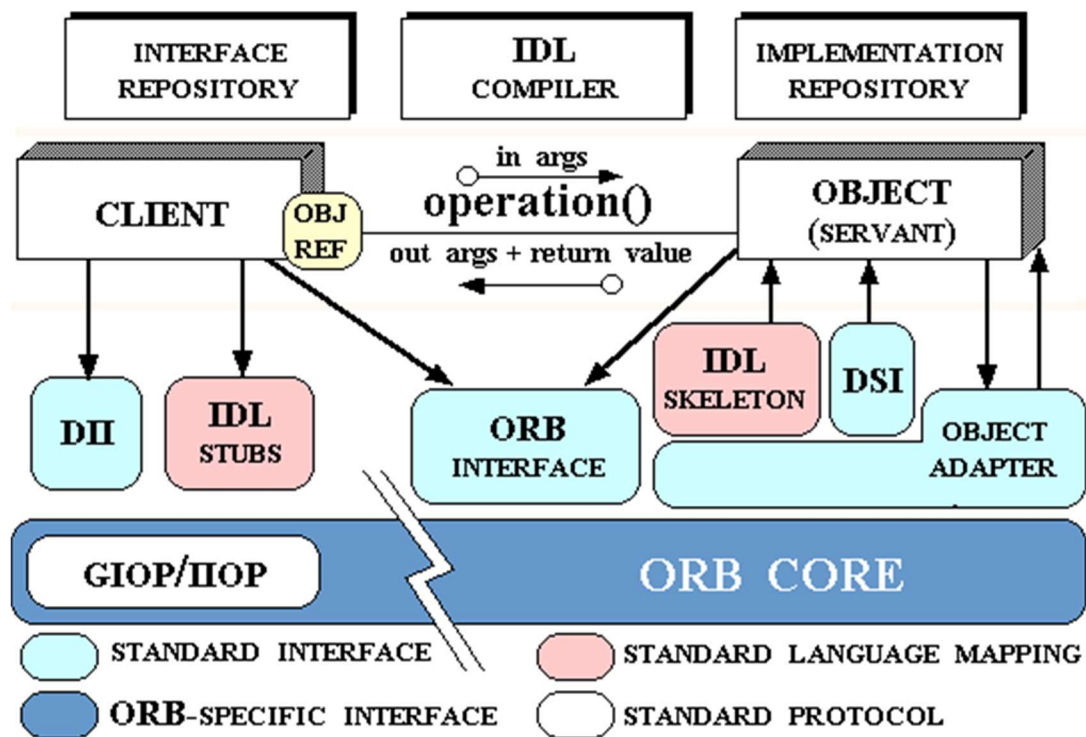


Figure 1.2: ORB architecture (Ref. professor's PPT)

CORBA enables communication between software written in different languages and running on different computers. Implementation details from specific operating systems, programming languages, and hardware platforms are all removed in CORBA. CORBA normalizes the method-call semantics between application objects residing either in the same address-space (application) or in remote address-spaces (same host, or remote host on a network).

CORBA uses an interface definition language (IDL) to specify the interfaces that objects present to the outer world. CORBA then specifies a mapping from IDL to a specific implementation language like C++ or Java.

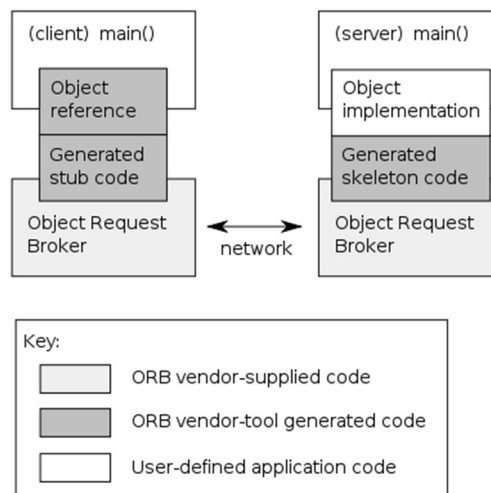


Figure 1.3: how the generated code is used within the CORBA infrastructure (Ref. Wikipedia)

## ❖ Distributed System Techniques & Data Structures

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

To store the players' information in the servers, I have used a 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 latter of player's username and it would store an Array list of players' HashMap as a value whose first latter of the username is same as Key. The structure of players' HashMap is as shown as below.

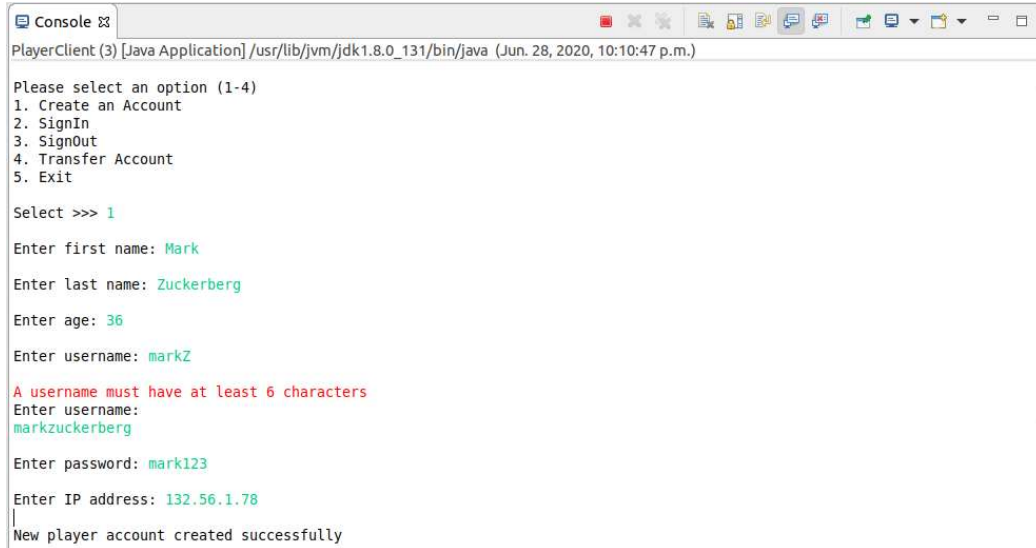
```
{
    "username": "Vasu123",
    "password": "Vasu1997",
    "firstname": "Vasu",
    "lastname": "Ratanpara",
    "age": 23,
    "ipaddress": "182.45.12.3",
    "status": "offline",
}
```

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



```

Console [3] [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 28, 2020, 10:10:47 p.m.)

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

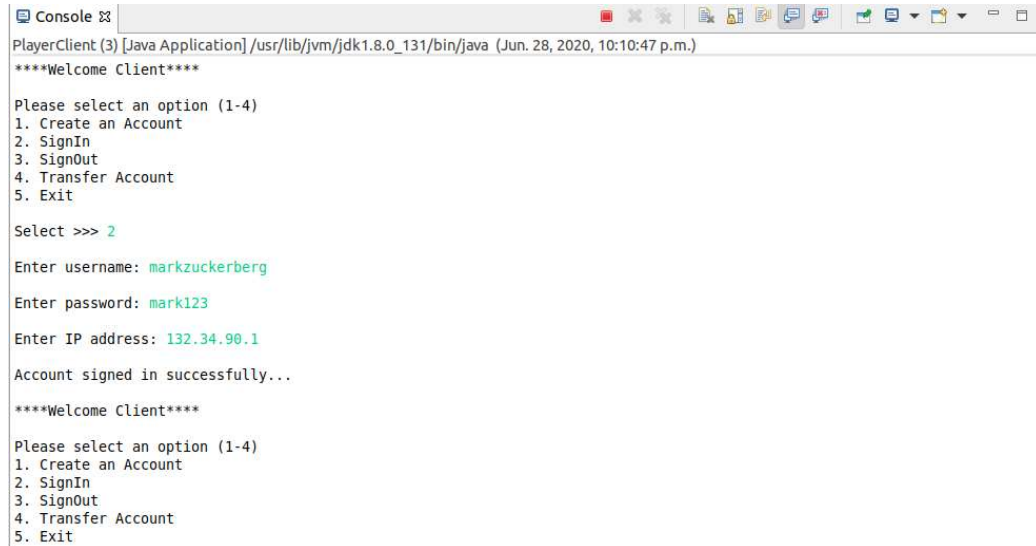
Select >>> 1

Enter first name: Mark
Enter last name: Zuckerberg
Enter age: 36
Enter username: markZ
A username must have at least 6 characters
Enter username: markzuckerberg
Enter password: mark123
Enter IP address: 132.56.1.78
New player account created successfully
  
```

Figure 1.3: Create an account

Here as you can see a player can register a new account with PlayerClient program. All fields are mandatory and username and password field has length validations which will allow minimum length of the field as 6 characters. The user account will be stored on the server which is relevant to given IP address. If the user is already present with same username then it will give an error will not create an account.

## 2. Sign in an account



```

Console [3] [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 28, 2020, 10:10:47 p.m.)
****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: markzuckerberg
Enter password: mark123
Enter IP address: 132.34.90.1

Account signed in successfully...

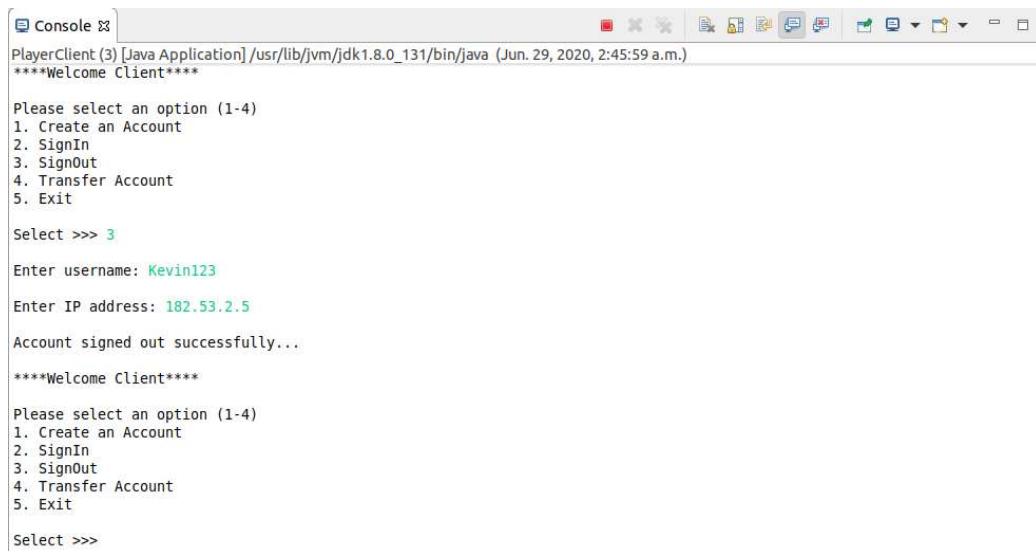
****Welcome Client****

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

Figure 1.4: 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 an 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 server.

## 3. Sign out an account



```

Console [3] [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 29, 2020, 2:45:59 a.m.)
****Welcome Client****

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

Select >>> 3

Enter username: Kevin123
Enter IP address: 182.53.2.5

Account signed out successfully...

****Welcome Client****

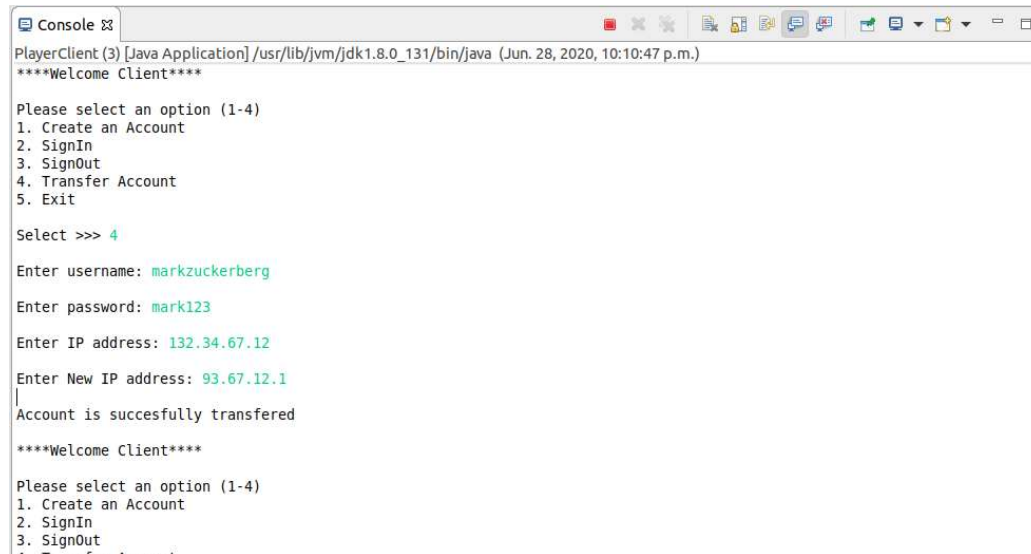
Please select an option (1-4)
1. Create an Account
2. SignIn
3. SignOut
4. Transfer Account
5. Exit

Select >>>
  
```

Figure 1.5: Sign out an account

Sign out operation is straight forward. 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



```

Console [3] [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 28, 2020, 10:10:47 p.m.)
****Welcome Client****

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

Select >>> 4

Enter username: markzuckerberg
Enter password: mark123
Enter IP address: 132.34.67.12
Enter New IP address: 93.67.12.1
Account is succesfully transfered

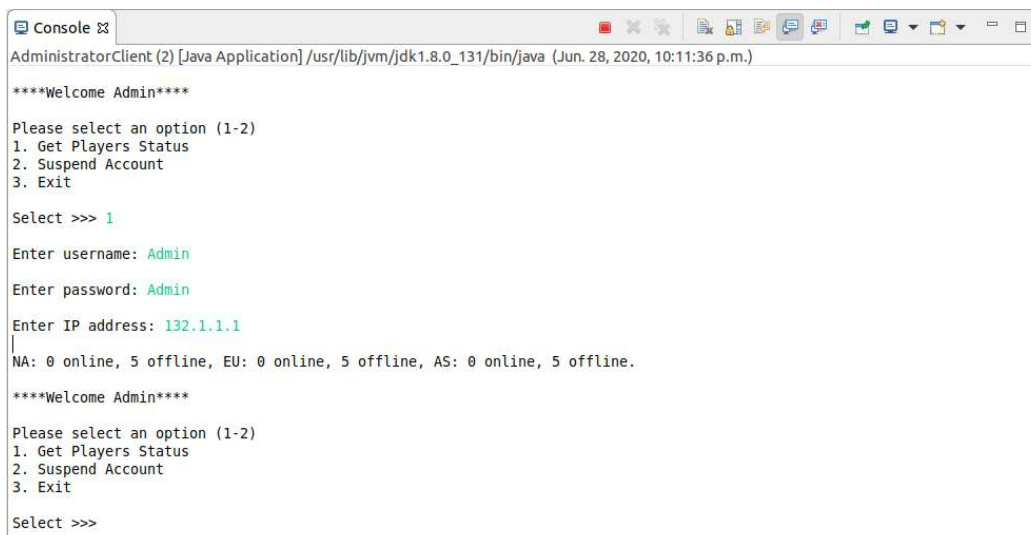
****Welcome Client****

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

Figure 1.6: 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 transfer from and old server to new server (both will be detected by each IP separately). The whole process is explained in-depth under an atomicity section.

#### 5. Get players' status



```

Console [2] [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 28, 2020, 10:11:36 p.m.)
****Welcome Admin****

Please select an option (1-2)
1. Get Players Status
2. Suspend Account
3. Exit

Select >>> 1

Enter username: Admin
Enter password: Admin
Enter IP address: 132.1.1.1
NA: 0 online, 5 offline, EU: 0 online, 5 offline, AS: 0 online, 5 offline.

****Welcome Admin****

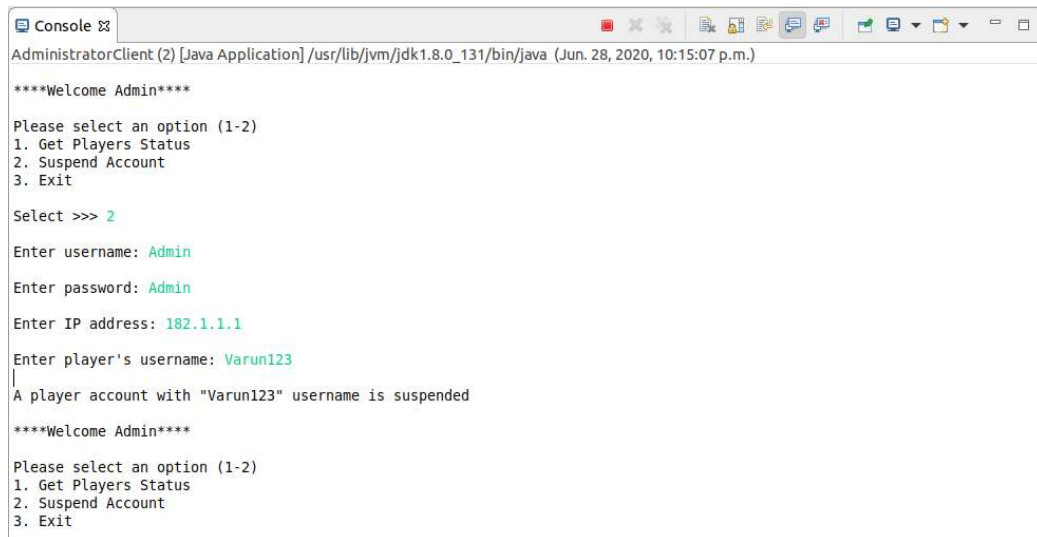
Please select an option (1-2)
1. Get Players Status
2. Suspend Account
3. Exit

Select >>>
  
```

Figure 1.7: 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 and UDP requests to other two server to know their players' status. Once it will get responses it will merge the response and send to an AdminClient.

## 6. Suspend an account



```

AdministratorClient (2) [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 28, 2020, 10:15:07 p.m.)

****Welcome Admin****

Please select an option (1-2)
1. Get Players Status
2. Suspend Account
3. Exit

Select >>> 2

Enter username: Admin
Enter password: Admin
Enter IP address: 182.1.1.1
Enter player's username: Varun123
A player account with "Varun123" username is suspended

****Welcome Admin****

Please select an option (1-2)
1. Get Players Status
2. Suspend Account
3. Exit

```

Figure 1.8.: Suspend an account

The suspend account will also take an Admin's username and password. Moreover, it will also take and IP address of a server and the username which needs to suspend. The suspend account will delete the user with given username and send a response back to AdminClient. If the user doesn't present of the server with 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 Client package. Which will test concurrency in scenario 7,8 and 9<sup>th</sup>. 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 Thread.sleep() function. The output would be as shown above the account will be suspended and when other thread will try to transfer an account it will give an error.



```

<terminated> Test [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 28, 2020, 10:41:52 p.m. - 10:51:47 p.m.)
>>> Concurrency Test
Case 1 : suspendAccount() will run first and then transferAccount()
Case 2 : transferAccount() will run first and then suspendAccount()
Case 3 : transferAccount() and suspendAccount() will run together

Select CASE >>> 1
Enter username: Siddall123
Enter password: Siddall123
Enter IPAddress: 132.34.21.1
Enter New IPAddress: 93.12.23.8

Result of suspendAccount() :
A player account with "Siddall123" username is suspended

Result of transferAccount() :
A player doesn't exists with given username

Final State =>
NA: 0 online, 4 offline, EU: 0 online, 6 offline, AS: 0 online, 4 offline.

```

Figure 1.9: Concurrency Test (Case 1)

## 8. Concurrency test (Case 2)

```

<terminated> Test [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 28, 2020, 10:59:08 p.m. - 10:59:53 p.m.)
>>> Concurrency Test
Case 1 : suspendAccount() will run first and then transferAccount()
Case 2 : transferAccount() will run first and then suspendAccount()
Case 3 : transferAccount() and suspendAccount() will run together

Select CASE >>> 2
Enter username: Gyles123
Enter password: Gyles123
Enter IPAddress: 93.12.1.67
Enter New IPAddress: 182.65.89.23

Result of transferAccount() :
Account is successfully transferred

Result of suspendAccount() :
A player doesn't exists with given username

Final State =>
EU: 0 online, 5 offline, AS: 0 online, 5 offline, NA: 0 online, 4 offline.

```

Figure 2.0: 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 very special case for concurrency testing. I removed synchronized keyword from transferAccount() and suspendAccount() methods from all 3 servers to see what will happen if the server gets two requests at same time and how it will handle by server.

```

Console [Java Application] /usr/lib/jvm/jdk1.8.0_131/bin/java (Jun. 29, 2020, 12:40:17 a.m. - 12:40:34 a.m.)
>>> Concurrency Test
Case 1 : suspendAccount() will run first and then transferAccount()
Case 2 : transferAccount() will run first and then suspendAccount()
Case 3 : transferAccount() and suspendAccount() will run together

Select CASE >>> 3

Enter username: Varun123
Enter password: Varun123
Enter IPAddress: 182.45.3.2
Enter New IPAddress: 93.56.7.8

Result of suspendAccount() :
A player account with "Varun123" username is suspended

Result of transferAccount() :
Something went wrong during account transfer rollback started...
Rollback successfully finished...

Final State =>
AS: 0 online, 4 offline, EU: 0 online, 5 offline, NA: 0 online, 5 offline.

```

Figure 2.1: Concurrency Test (Case 3)

As you can see in the screenshot, the 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 rollback will be successful after that.

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

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

- IP validation

```

****Welcome Admin****

Please select an option (1-2)
1. Get Players Status
2. Suspend Account
3. Exit

Select >>> 2

Enter username: Admin
Enter password: Admin
Enter IP address: 163.56.3.2

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 2.2: IP validations

- Age, Username and Password validations

```

Enter first name: dfd
Enter last name: ds
Enter age: dzfd
Enter age:
Please input Integer only
34

Enter username: sdfs

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

Enter password: ds

A password must have at least six characters
Enter password:
dfdsfdfdf

```

Figure 2.3: Age, Username and Password validations

## ❖ important/Difficult Part in The Assignment

### 1. Atomicity

#### Atomicity in transferAccount( )

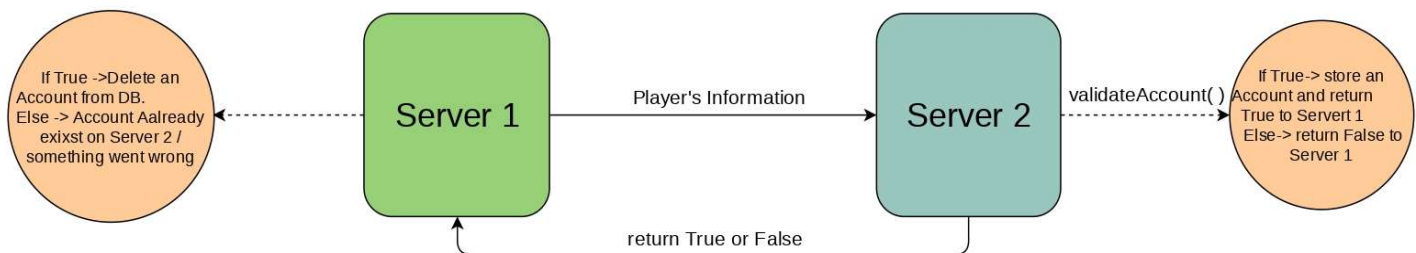


Figure 2.4: Atomicity in transferAccount()

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

When the server 1 get 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 an 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 same name is present then the server 2 will send "False" to server 1 and server 1 will send a message to client that the user is already exist 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 the any reasons 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. HashMap in Java

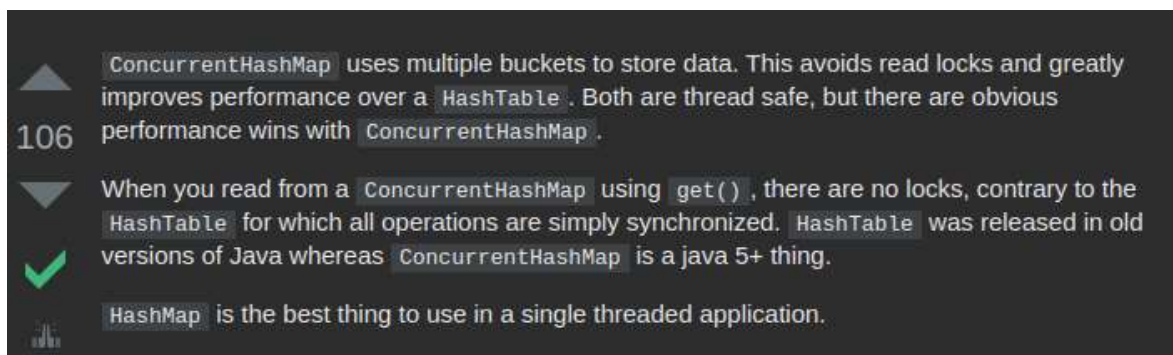


Figure 2.5: ConcurrentHashMap vs HashMap (Ref. [Stack Overflow](#))

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

## 3. A Special Method: getConfig()

To reduce the run time complications, I have created a new method called getConfig() in each server files. which will pass the arguments for ORB.int() method which is passing statically from the getConfig() method. The ORB port is defined in ORB\_PORT which is 1050.

## ❖ Distributed System UML Design

### ➤ Full UML Diagram

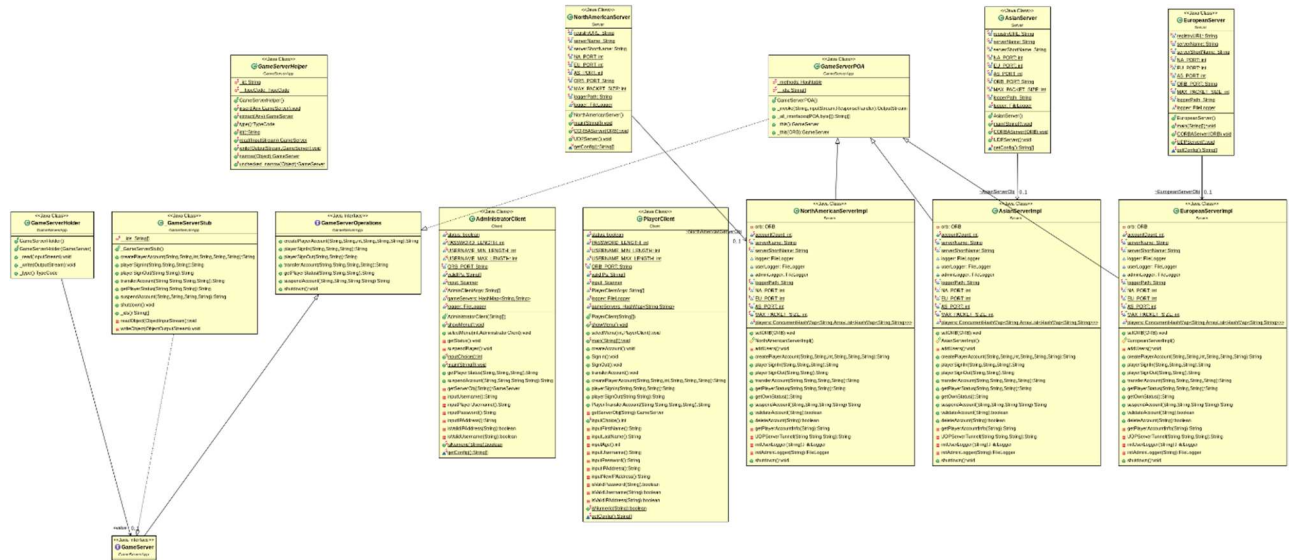


Figure 2.6: Full UML diagram

The figure 2.6 and 2.7 represent the UML class diagrams. In figure 2.6 all classes are covered, and it represent the different relation between all classes.

The GameServer interface and abstract class GameServerPOA extends GameServerOperations interface. The class \_GameServerStub implements the GameServer class. The GameServerOperation interface will have all abstract method which will be overwrite by the server.

At the CORBA server end the AsianServerImpl, EuropeanServerImpl and NorthAmericanServerImpl extends the GameServePOA class. Moreover, the Objects of each of them will be created in their separate server files. i.e. AsianServer

At the Client side only two classes GameServer and GameServerHelper are used in all client implementations.

In second class diagram it represent the in-detailed structure of each server classes. Each server will have 2 files where 1 file would work as a UDP and CORBA server and other will extends the required class and override the abstract methods so it can be use in server files.

Lastly, the CORBA server and UDP server will be run separately from single file via threads so they cannot block each other.



## ➤ Servers' UML Diagram

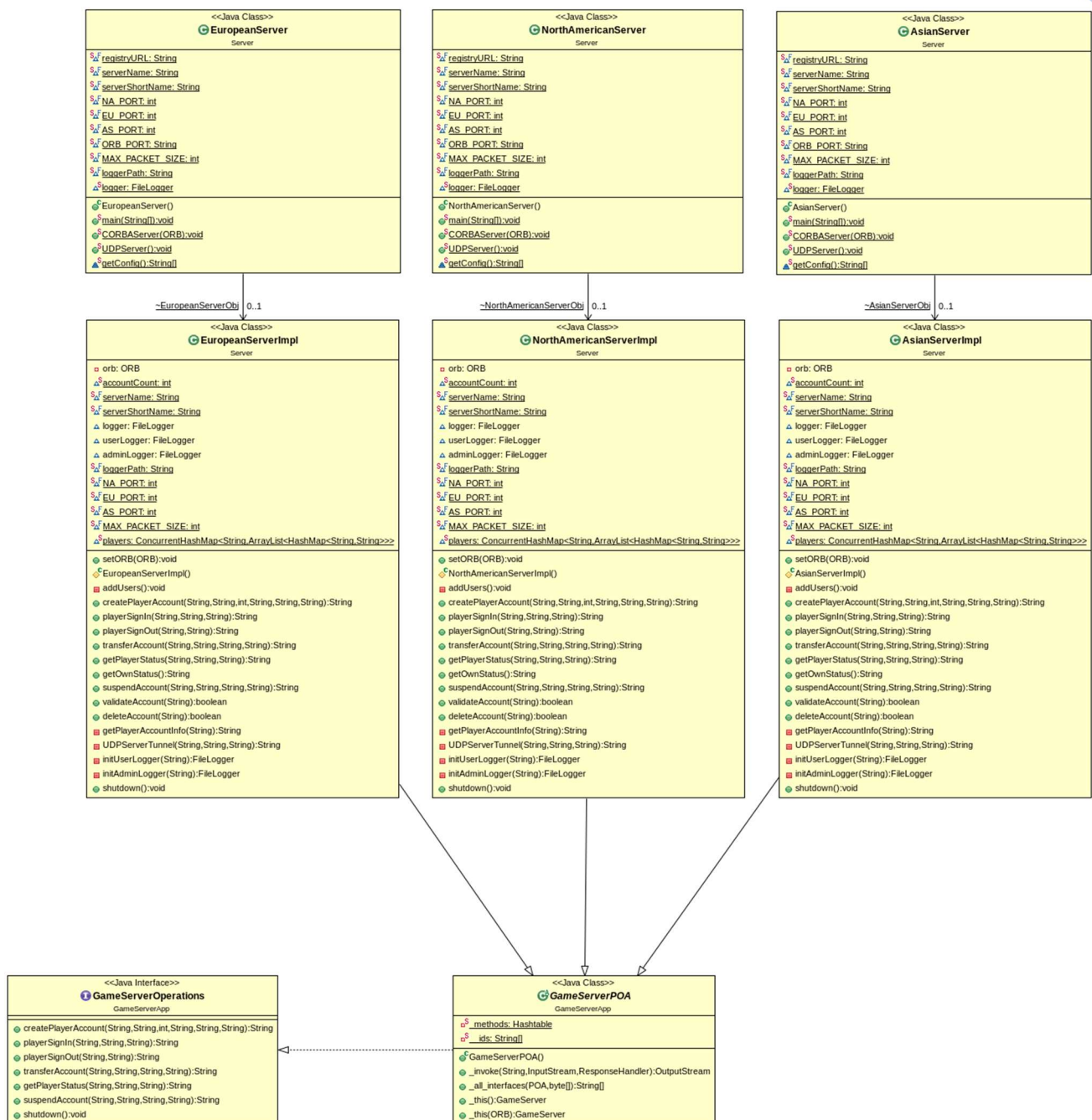


Figure 2.7: Servers' UML diagram

**Note:** To see the diagrams properly please go to uml folder which is inside on Assignment 2 folder.

## ❖ Project's Defaults

### ➤ ORB Port

ORB\_PORT = 1050

### ➤ UDP Server Ports

NA\_PORT = 5001

EU\_PORT = 5002

AS\_PORT = 5003

**Note:** Please kill all services which are running on 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 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 2" folder. Put "Assignment 2" folder in your Eclipse workspace and open the project in your Eclipse.

There are 4 packages in the folder namely Client, GameServerApp, Server and Logger.



Figure 2.8: Assignment 2 packages

## ❖ src folder

1. Client package
  - AdministratorClient.java
  - PlayerClient.java
2. GameServerApp (Auto generated CORBA java files)
3. Server package
  - AsianServer.java (Asian Server)
  - AsianServerImpl.java
  - EuropeanServer.java (European Server)
  - EuropeanServerImpl.java
  - NorthAmericanServer.java (North American Server)
  - NorthAmericanServerImpl.java
4. Logger package
  - FileLogger.java



## ❖ logs folder

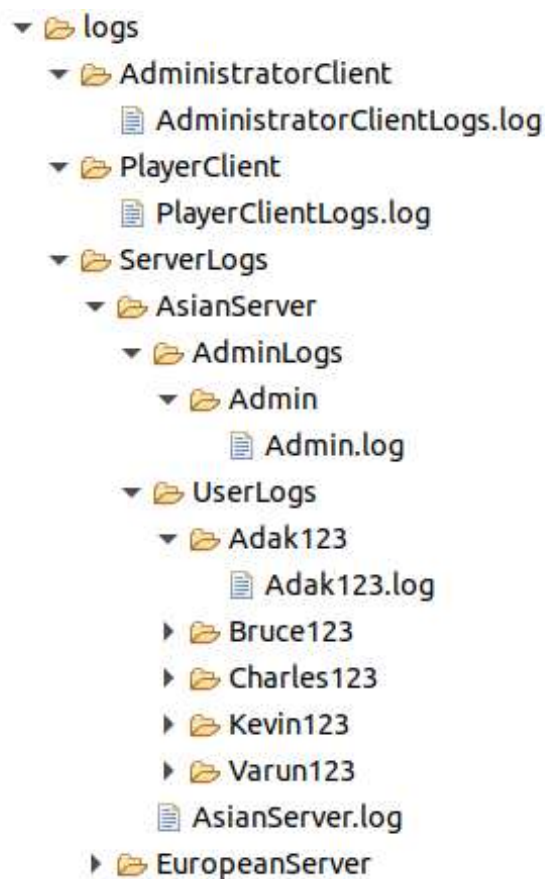


Figure 2.9: Log Folder Structure

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)

## ❖ uml folder

The Both UML diagrams are stored inside the uml folder which is located in Assignment 2 folder.

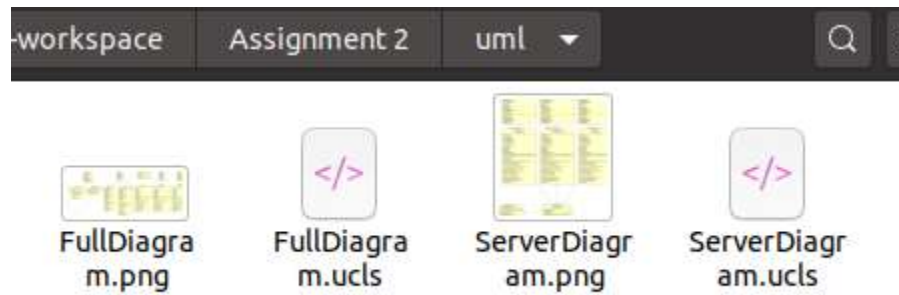


Figure 3.0: UML diagrams

## ❖ Starting ORBD Server in Eclipse

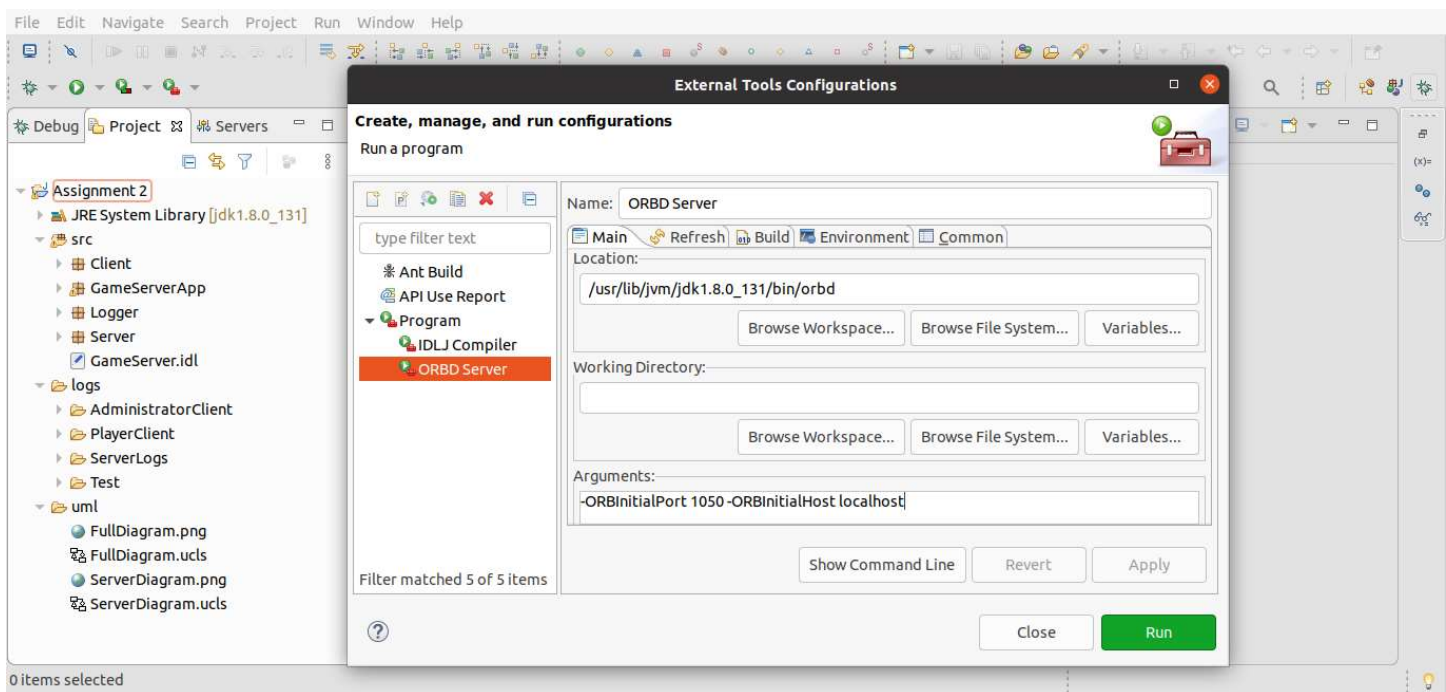


Figure 3.1: starting ORBD server inside Eclipse

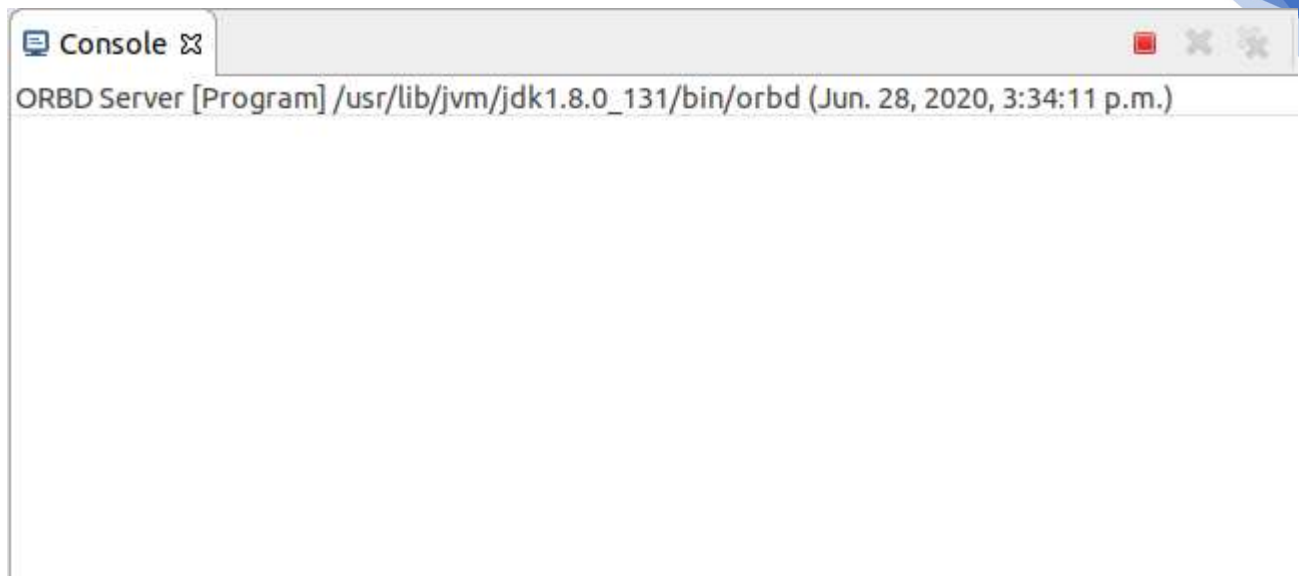


Figure 3.2: ORBD server has been started in Eclipse

OR

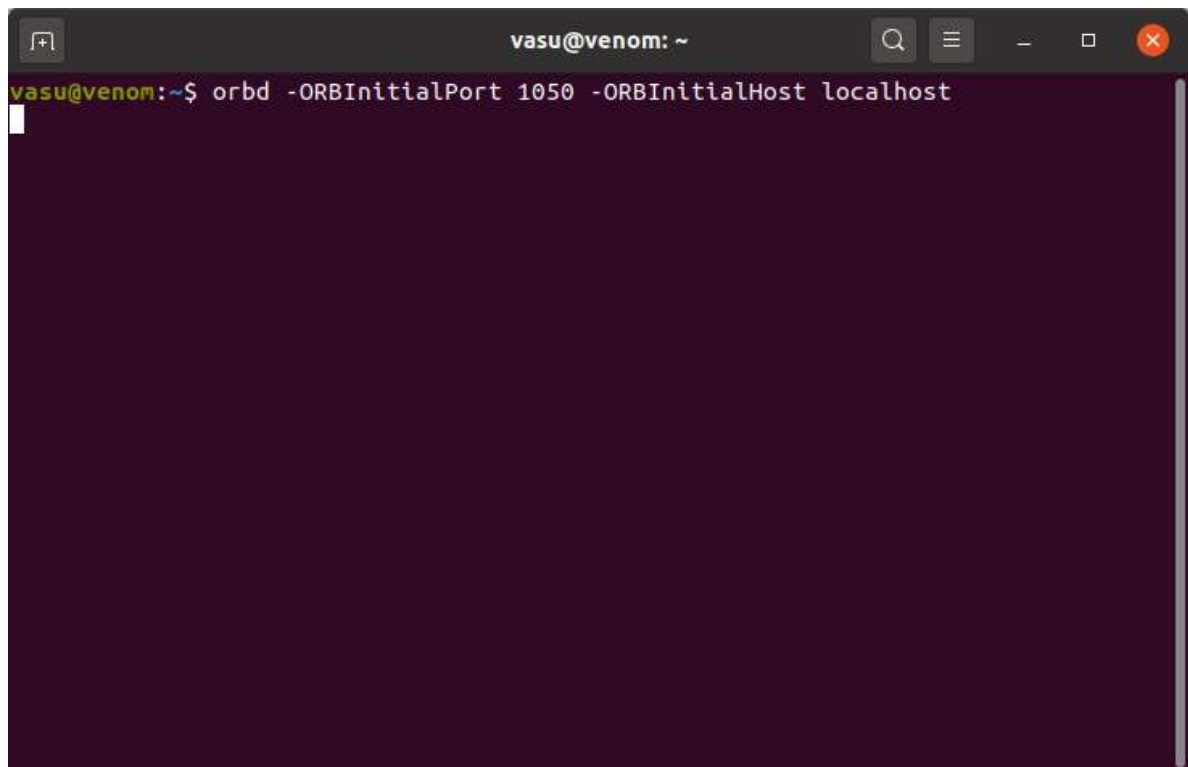


Figure 3.3: Running ORBD outside of the Eclipse

## ❖ Starting Asian Server

There will be no requirement to pass the arguments at the run time because it is passed statically with getConfig() for all of 3 servers to ORB.init().

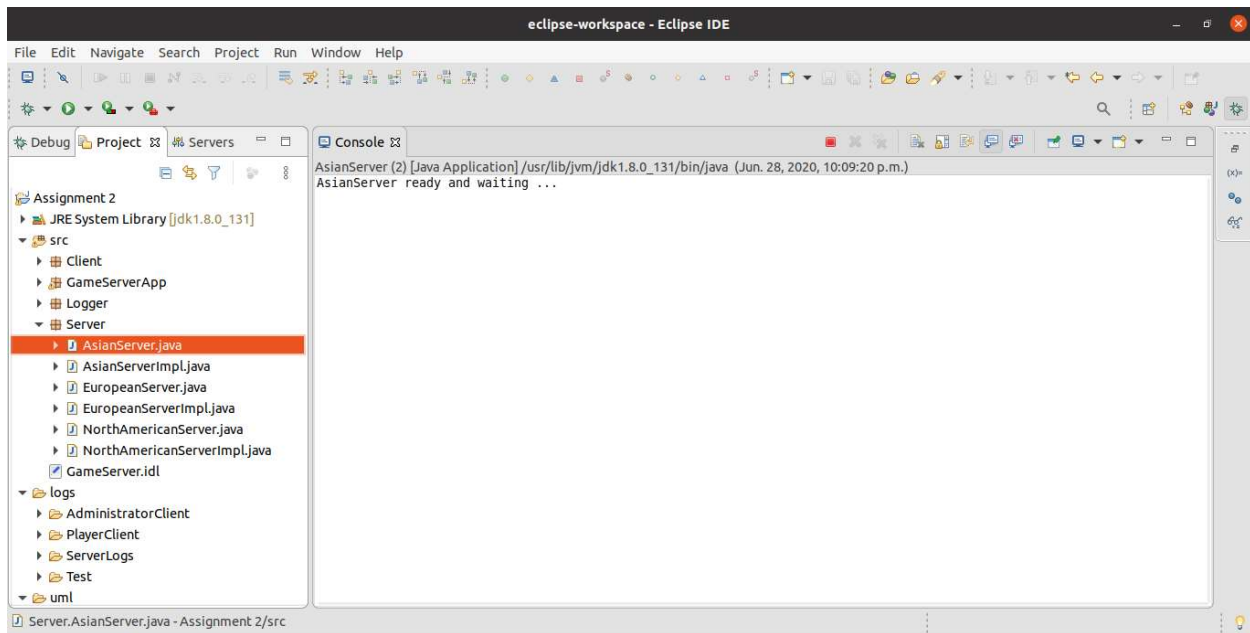


Figure 3.4: Starting the Asian server

## ❖ Starting European Server

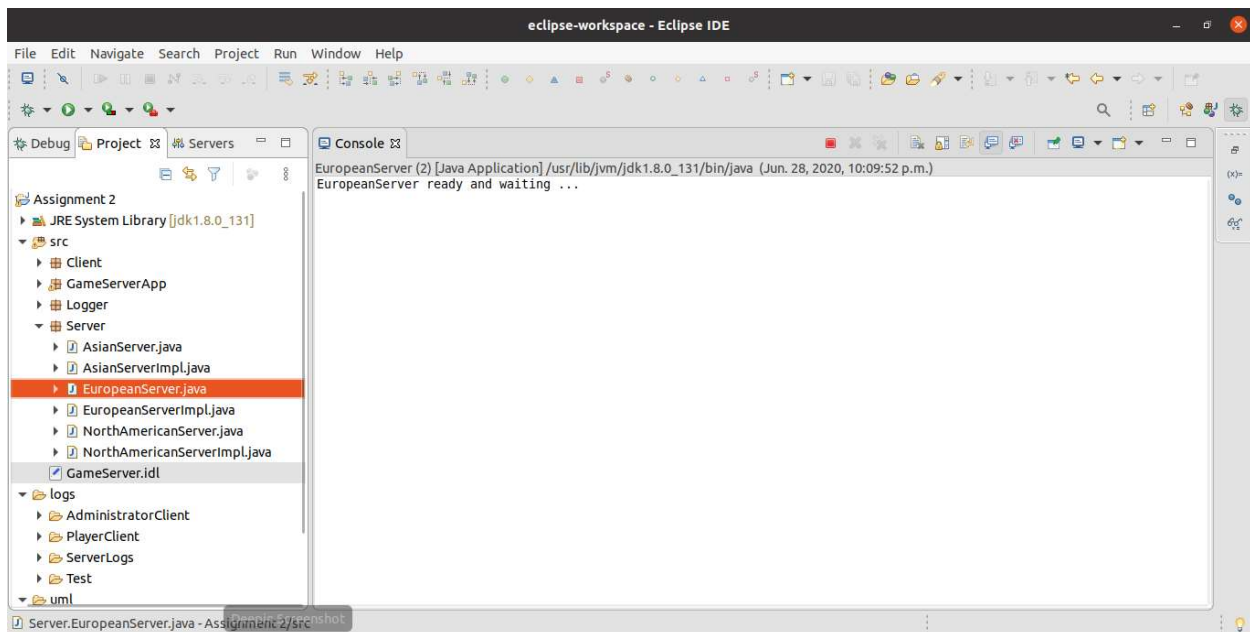


Figure 3.5: Starting the European server

## ❖ Starting North American Server

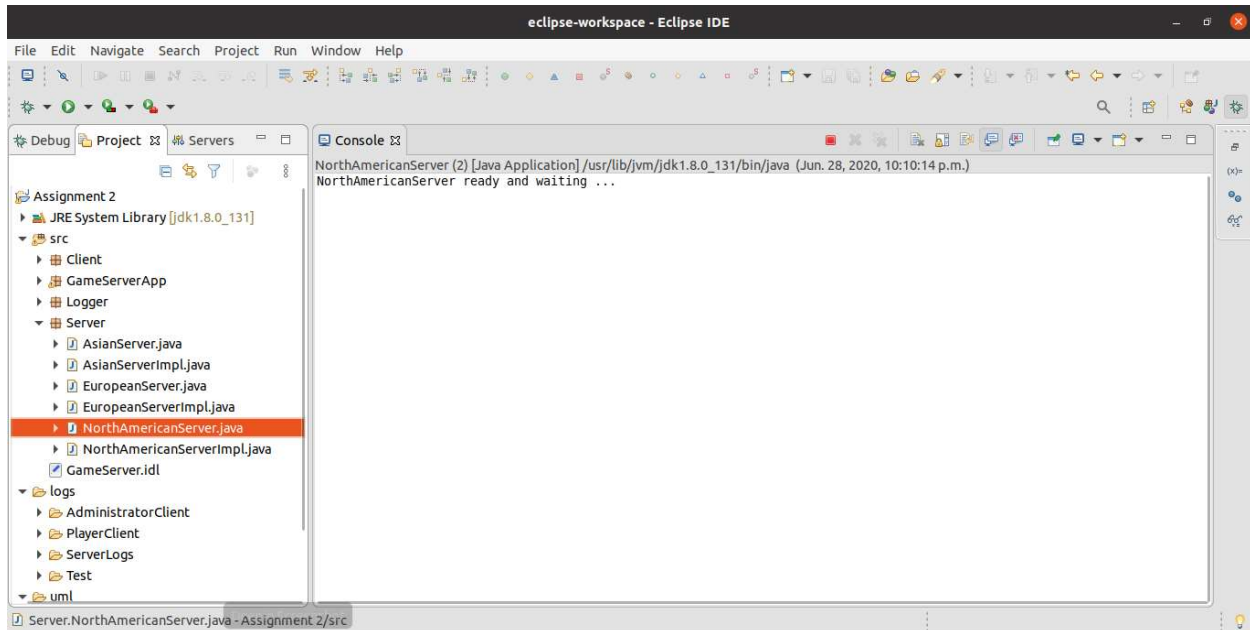


Figure 3.6: Starting the North American server

## ❖ Starting PlayerClient

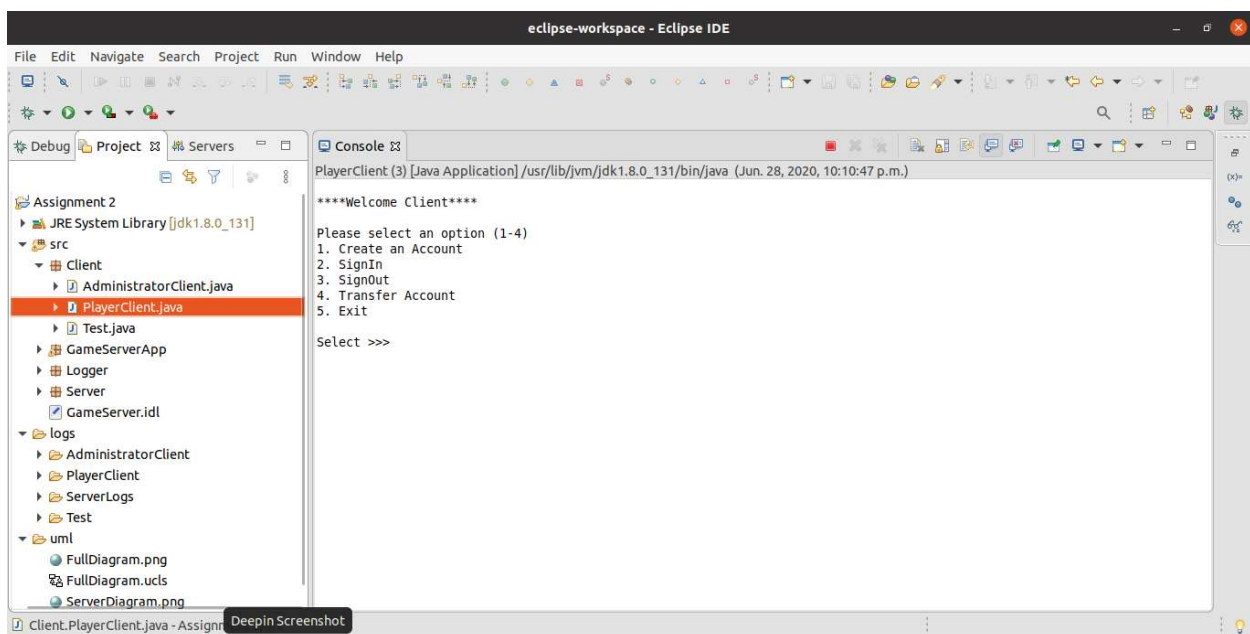


Figure 3.7: Starting the PlayerClient

## ❖ Starting AdministratorClient

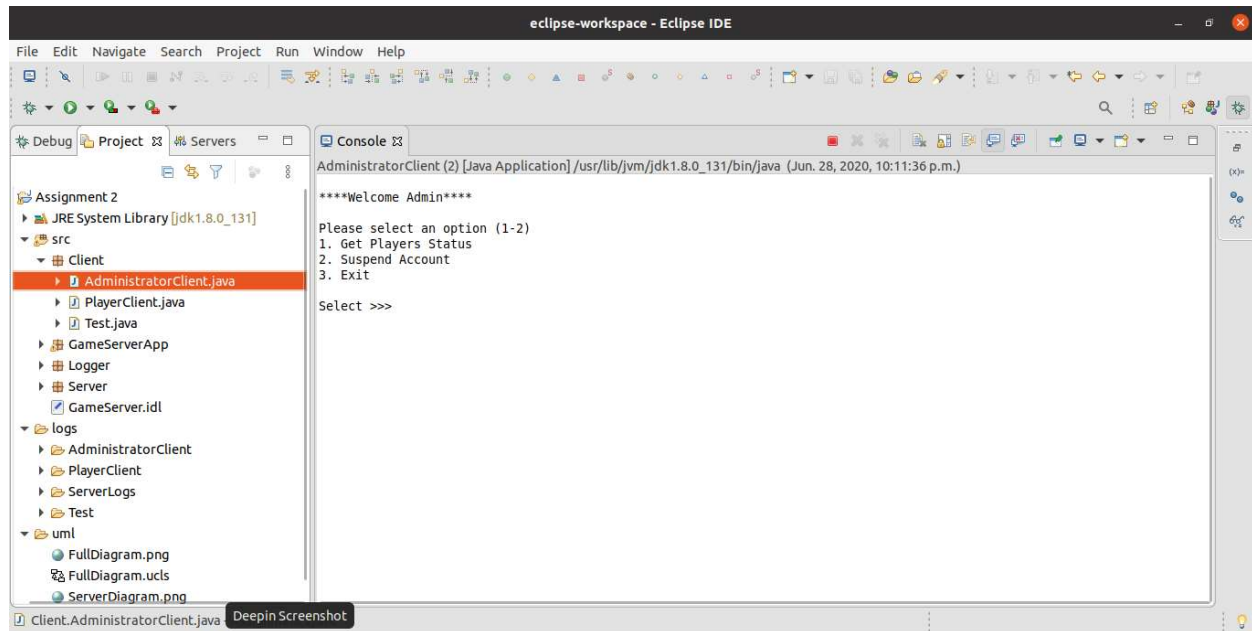


Figure 3.8: Starting the Admin Client

## ❖ Logs

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

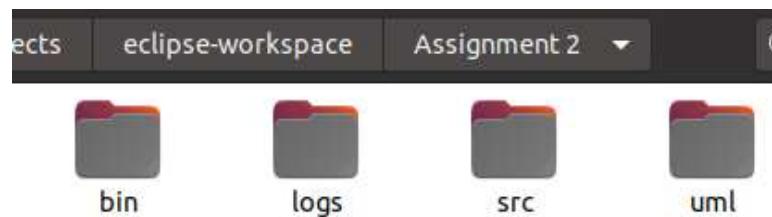


Figure 3.9: Assignment 2 folder structure

There are 3 folders inside “logs” folder

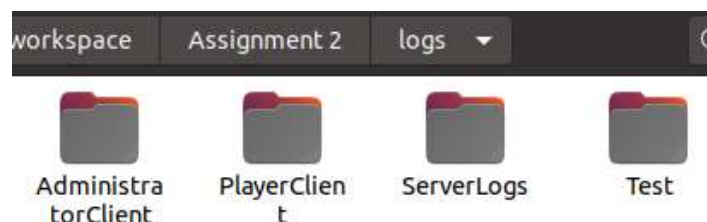


Figure 4.0: Logs folder structure

Inside “ServerLogs” folder there is sperate folder for each server which contains the logs for individual servers and their users.

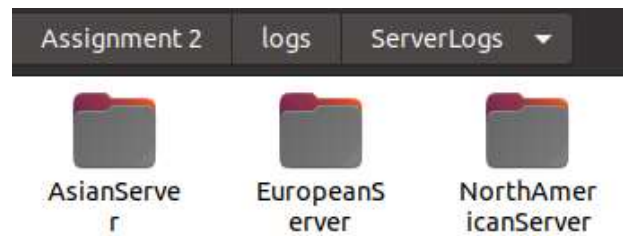


Figure 4.1: ServerLogs folder structure

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 4.2: AsianServer folder structure

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



Figure 4.3: UserLogs folder structure

User log for 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](mailto:vasu.ratanpara@mail.concordia.ca)

# Thank You