

**COMP6231- Distributed System Design**

**Assigment-3**

Distributed Player Status System (DPSS) using Java Webservices

**Date:** 12 July 2020.

**Recipient:** Prof. Mohammed Taleb

**Student Name:** Raj Mistry (40119206)

* **Introduction:**

The main objective of this assignment is to design and implement a Distributed Player Status System as a web service which manages player’s status across multiple game servers using Java Webservices. In addition to this we have to make our application properly synchronized with the use of multi-threading in order to handle concurrent requests with ease. User Datagram Protocol is used by each server to communicate with other servers, sending and receiving requests to perform inter-server operations.

* **Web Service:**

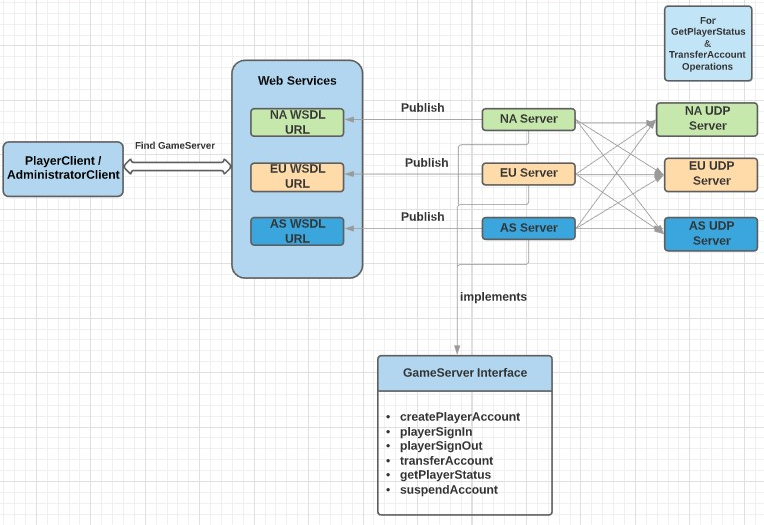
Web Service is a collection of standards and open protocols that are used for communication between devices or applications. Applications written in different programming language and running on different OS can use web services to communicate and exchange information with each other over the network. Web service provides server interface to client which enables the client to communicate with sever. Client uses this interface operations and send request to server and get response for that particular request through standardized XML messaging system which is usually transmitted over HTTP. All the standard web services are worked using these common components: SOAP (Simple Object Access Protocol) , UDDI (Universal Description, Discovery and Integration) & WSDL (Web Services Description Language).

Types of web services:

1. SOAP Web Services
2. RESTful (Representational State Transfer) Web Services

In this assignment I have implemented first type of web service by using JAX-WS.

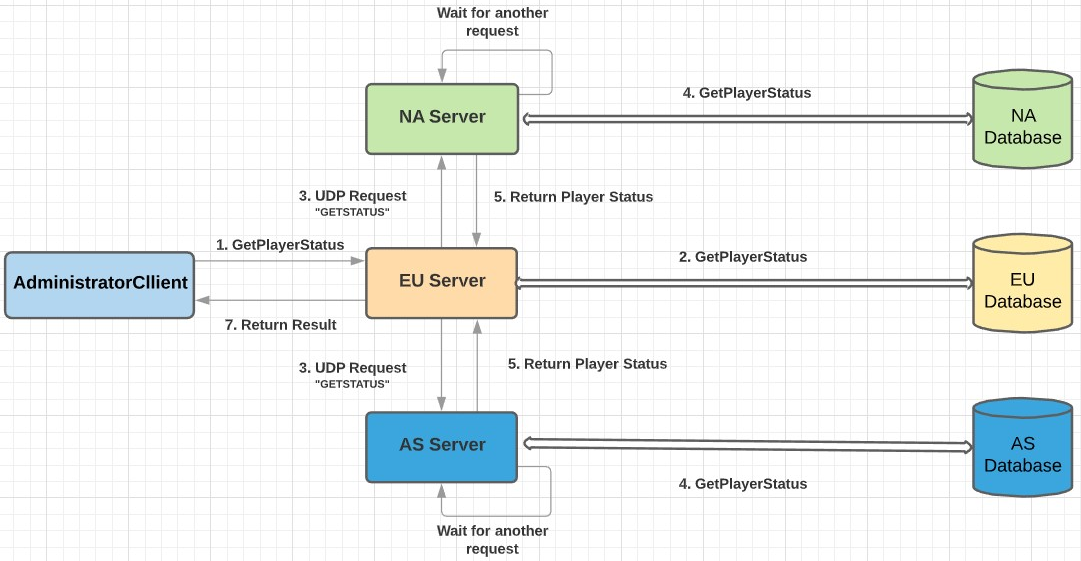
* **Design Architecture:**



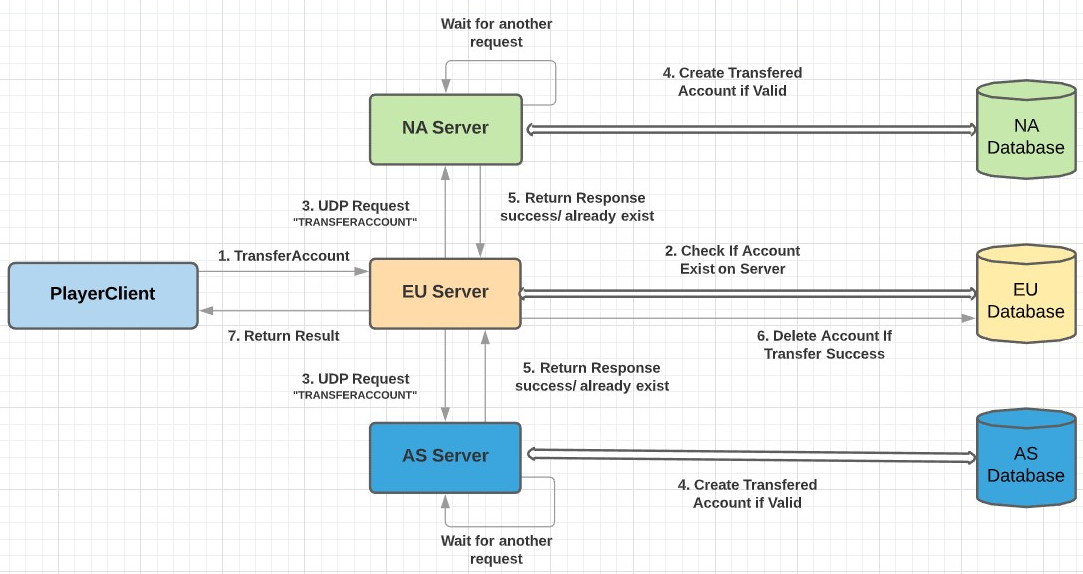
* **UDP:**

The User datagram Protocol is a transport layer protocol. It is connectionless and unreliable protocol. To communicate using UDP it is not required to establish connection prior to data transfer. A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The Datagram Packet and Datagram Socket classes in the java.net package implement datagram communication using UDP.

* In this assignment I have implemented UDP for communication among three servers (NA, EU and AS) in order to perform operations getPlayerStatus() and transferAccount(). Each server Creates its own UDP thread when server starts.
* In getPlayerStatus () the current server communicates with other servers using UDP request “GETSTATUS” to fetch the player accounts status on corresponding server and returns server specific online and offline player accounts count. (In this UDP request will be sent to rest of the 2 servers)



* In transferAccount() checks the HashMap for the required player account. If found, the server transfers the account to specified remote server by sending UDP request “TRANSFERACCOUNT” and after successfully transferring the record to remote server, it will remove the record from its storage and then server informs the client that the player account transfer is successful.(In this UDP request will be sent to only 1 server where we want to transfer Player Account[Destination Server] e.g. To transfer Account from EU to AS, UDP Request will be sent to AS only not to NA])



* **Multithreading and Synchronization:**

Thread is a lightweight process which has its own stack. When Multiple threads try to access the same resources and produces wrong result because of data corruption. This happens because while one thread is accessing resource same time other thread interferes and access it and now both perform operation on that old value and write their result which may produce wrong result.

Synchronization is a procedure in which is used to avoid interference with the other threads and memory consistent errors. The keyword synchronized is used to implement Competition Synchronization which means while one thread is executing the critical region where shared resource is used all the other threads will wait until its execution of critical region finished.

There are Two types of synchronizations used in java:

* Method Synchronization
* Block Synchronization.

***Synchronization Used in this Assignment:***

* Block level Synchronization:
  + A specific part or block of code that is needed to be synchronized is put under its own specific scope.
* Method level Synchronization
  + A specific methods or functions in java is synchronized using synchronize keyword before the method name.
* **Data Structure and Concurrency:**

Here in this assignment in order to improve concurrency I have used ConcurrentHashMap to as the storage for Player Accounts as it is an enhancement of HashMap, as we know that while dealing with threads in our application simple HashMap is not good choice because performance wise HashMap is not up to the mark. ConcurrentHashMap class is thread safe and it is good choice when we need high concurrency in our application. It does not throw a “ConcurrentModificationException” if one thread tries to modify it while another is iterating over it.

**Syntax:**

ConcurrentHashMap<String, List<Object>> accounts = new ConcurrentHashMap<String, List<Player>> ();

Where Player is the Model class which stores details of player account in its object.

The Fine-grained locking is used to put intrinsic lock while doing parallel access to data resources so that data is accessed as required without infecting the data. This increases the efficiency and maximizes the concurrency to perform concurrent operations.

Concurrency is implemented by using the synchronized(){}-block in java which puts temporary lock on the object or list passed to it as parameter in this implementation.

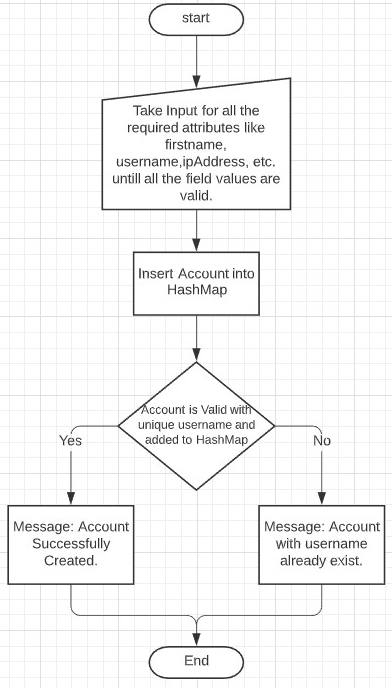
* **Operations:**

1. **Create Player Account.**

* While creating player account first all the attributes will be taken as input from user. During the input it will accept only valid inputs for particular field and if invalid input is given it will keep asking for valid input. The validations (All attributes value must be entered. It can’t be empty & without any whitespaces) for input are:

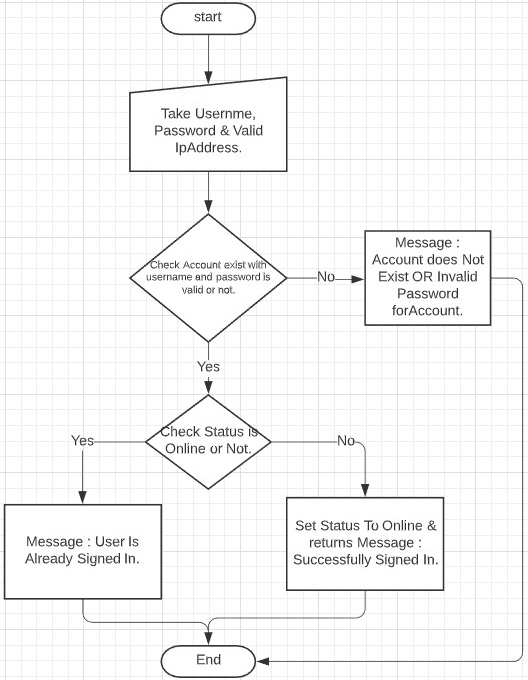
1. FirstName & LastName must contain only alphabets.
2. Age must be positive integer.
3. Username must start with alphabet & its length must be between 6 characters to 15 characters.
4. Password length must be minimum 6 characters.
5. IpAddress must be from valid ipV4 format & it should be from one of these range: 132.xxx.xxx.xxx, 182.xxx.xxx.xxx & 93.xxx.xxx.xxx.

* Once all the valid inputs are entered & Player Account with username does not already exist then it will be added to HashMap with Capital-letter of username’s first character as Key.
* Message will be return to user as well as it will be written in Log Files.

****

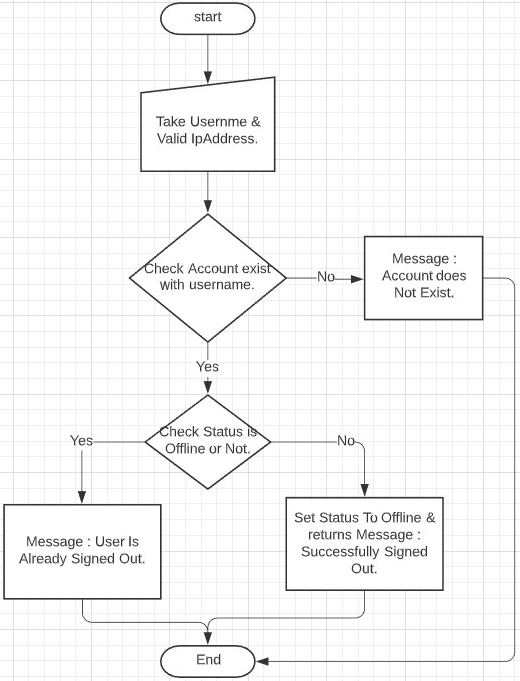
1. **Player Sign In.**

* While client want to sign in into player account program will take username, password and valid IpAddress as input. Then on the server first it will check if there exist a player account with entered username or not. If there is no such a player account with that username it will return message that player account not exist. Otherwise if player account exist it will verify password and if password is correct it will check status and set it online if it’s not already online and based on actions it will return success or error message. Message will be return to user as well as it will be written in Log Files.

****

1. **Player Sign Out.**

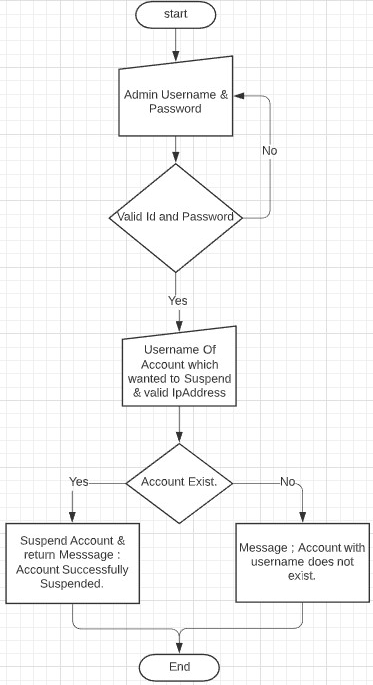
* While client want to sign out from player account program will take username and valid IpAddress as input. Then on the server first it will check if there exist a player account with entered username or not. If there is no such a player account with that username it will return message that player account not exist. Otherwise if player account exist it will check status and set it offline if it’s not already online and based on the actions it will return success message.
* Message will be return to user as well as it will be written in Log Files.

****

**(Note:** For Transfer Account and Get player status operations the way it works is described in UDP description above where it takes required inputs from user and verifies them and if all valid it will send particular UDP request to other server(s) based on operation. Then server will perform operation based on received request and returns result to server who sent request and then these results are returned to client and also written into log files.)

1. **Suspend Player Account.**

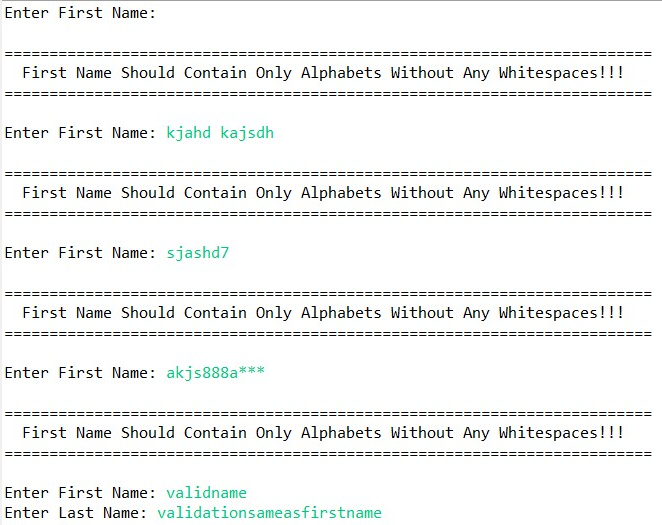
* While Administrator want to suspend player account program will verify Admin account credentials and if they are valid it will take username of player account which he wants to suspend and valid IpAddress as input. Then on the server first it will check if there exist a player account with entered username or not. If there is no such a player account with that username it will return message that player account not exist. Otherwise if player account exist it will remove it from HashMap and return success message. Message will be return to user as well as it will be written in Log Files.

****

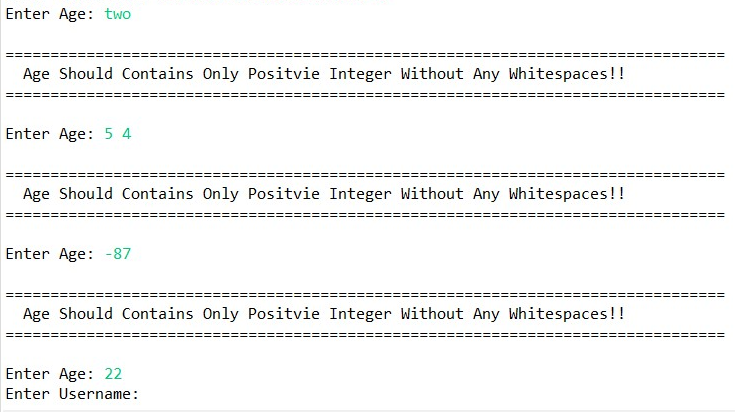
* **Test Cases:**

1. **Validations (All attributes must be entered & can’t be empty)**

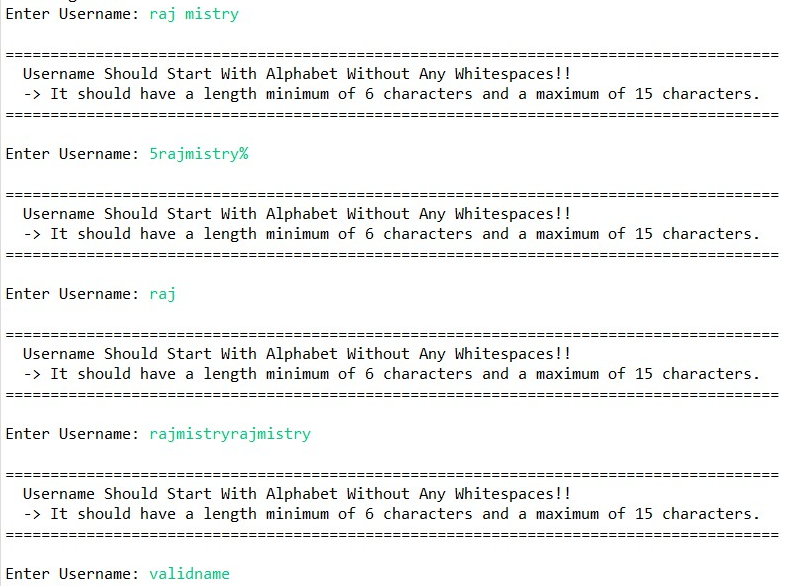
* **First Name** and **Last Name** Should Contain Only Alphabets without any Whitespaces.



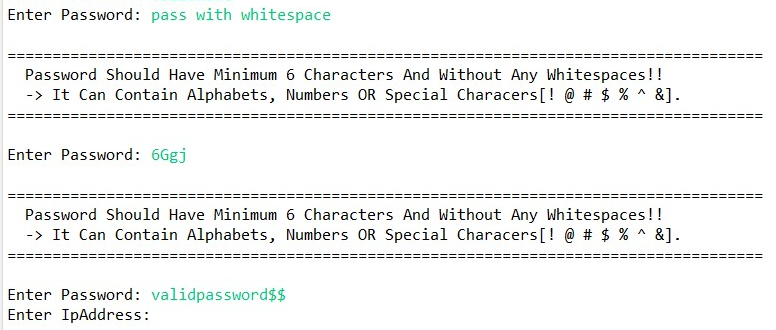
* **Age** must be Positive Integer without any whitespaces.



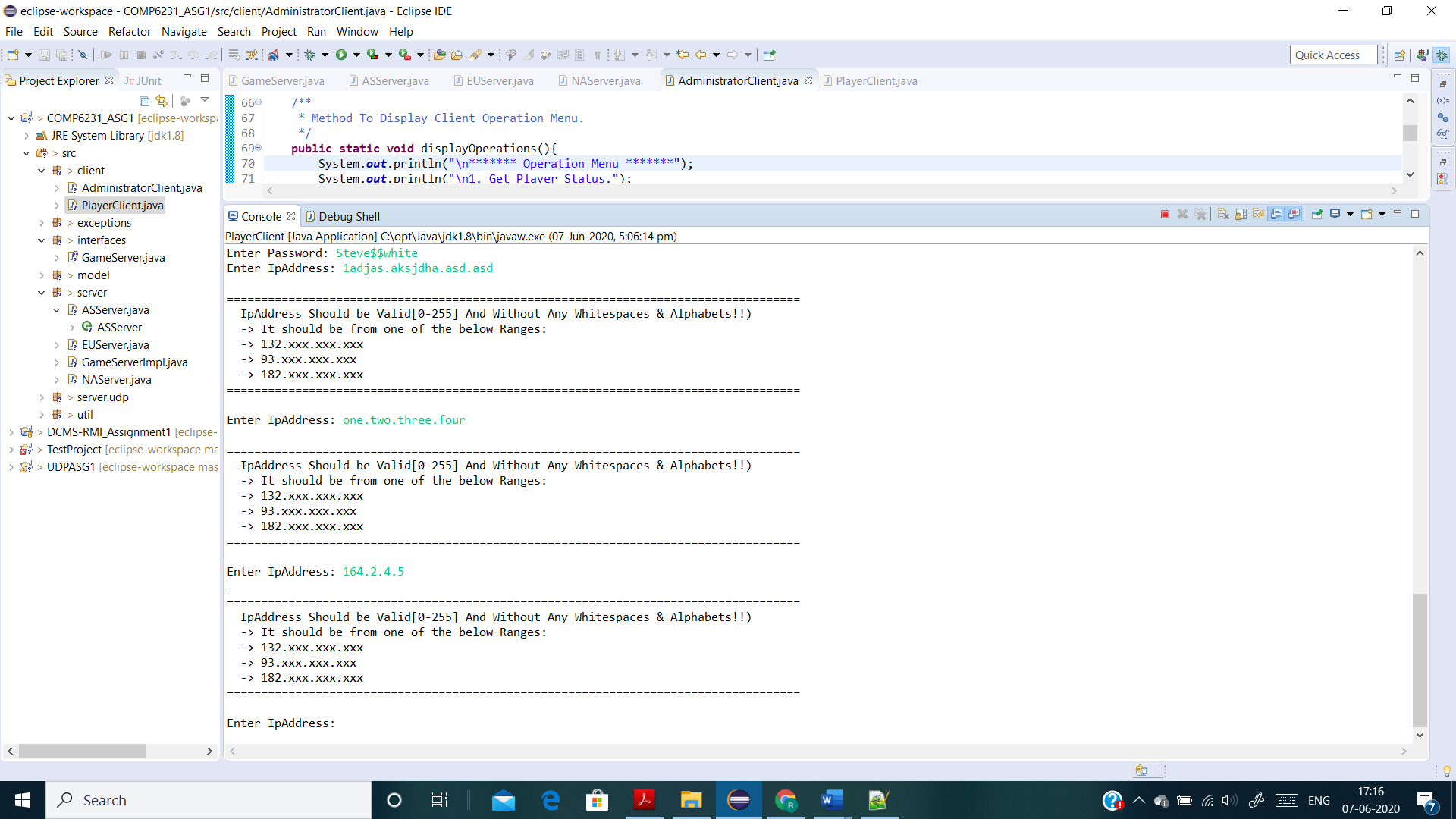
* **Username** must be start with alphabet and should not contain any whitespaces and length must be minimum of 6 and maximum of 15 characters.



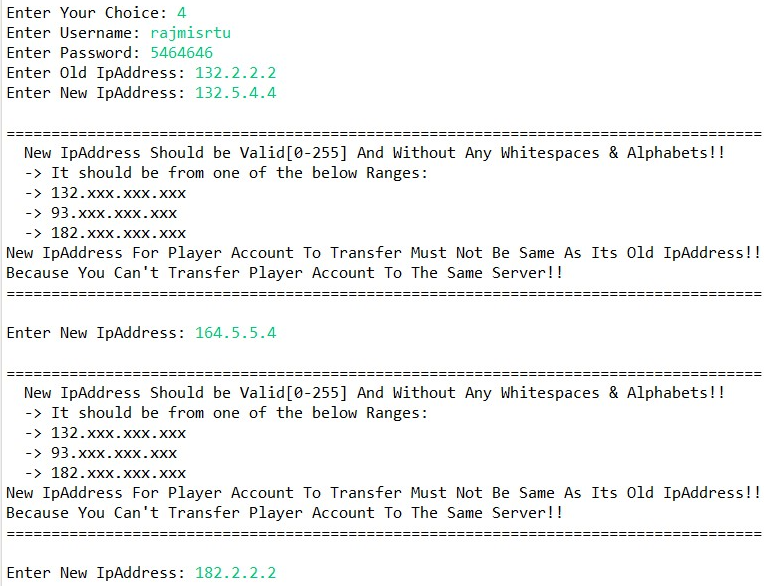
* **Password** must not contain any whitespaces and should have minimum length of 6 characters.



* **IP Address** must be entered as per valid IPv4 format & must be entered from one of the 3 ranges (132.xxx.xxx.xxx, 93.xxx.xxx.xxx, 182.xxx.xxx.xxx).



* **New IP Address for Transfer Account** must be valid Ip Address as above and it must be different from old Ip Address as Account cannot be transfer to same server.



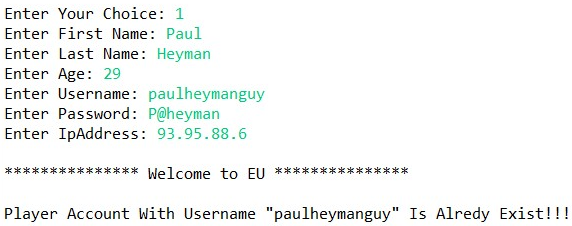
* **Username and Password for Administrator** must be “Admin” without any whitespaces and must be exact same as shown (Case Sensitive).



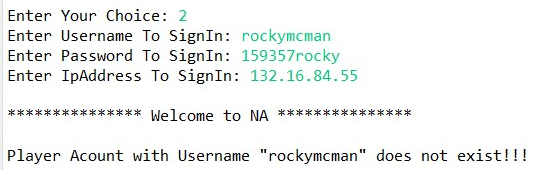
1. **Create New Player Account (Successfully Created New Player Account)**

****

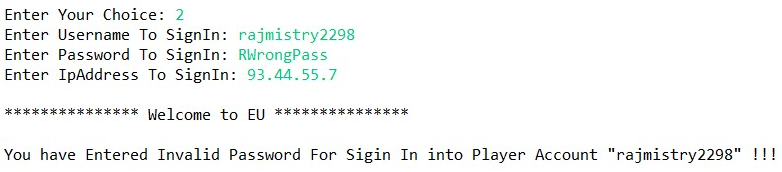
1. **Create New Player Account (Unsuccessful – Player Account with same username Already Exist)**

****

1. **Player Sign In (Unsuccessful - Player Account in which trying to Sign In Not Exist)**

****

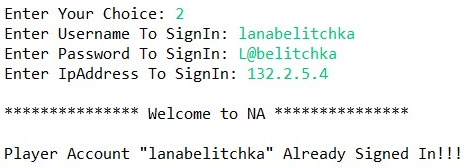
1. **Player Sign In (Unsuccessful - Wrong Password Entered For Player Account to Sign In)**

****

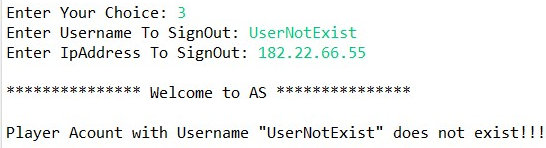
1. **Player Sign In (Successfully Signed in into Player Account)**

****

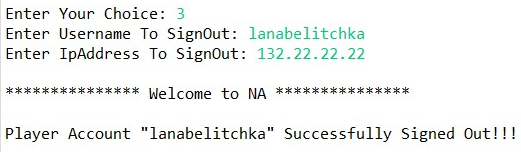
1. **Player Sign In (Trying to sign in into Player Account which is Already Signed In)**

****

1. **Player Sign Out (Unsuccessful - Player Account from which trying to Sign Out Not Exist)**

****

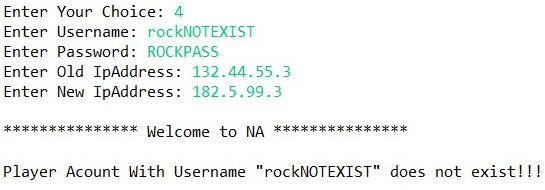
1. **Player Sign Out (Player Account Successfully Signed Out)**

****

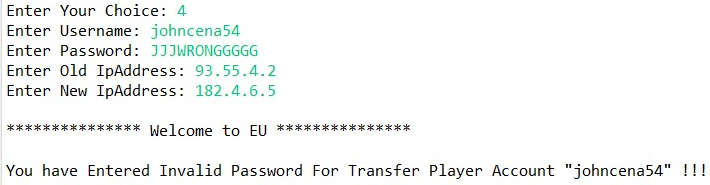
1. **Player Sign Out (Player Account Trying To Sign Out Which Is Already Signed Out)**

****

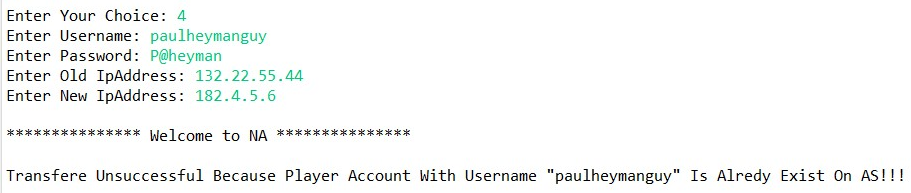
1. **Transfer Player Account (Unsuccessful – Player Account which client trying to transfer does not exist on server)**

****

1. **Transfer Player Account (Unsuccessful – Player Account which client trying to transfer is exist on server but wrong password is entered for Account)**

****

1. **Transfer Player Account (Unsuccessful – Player Account which client trying to transfer, Player Account with same username already exist on destination server)**

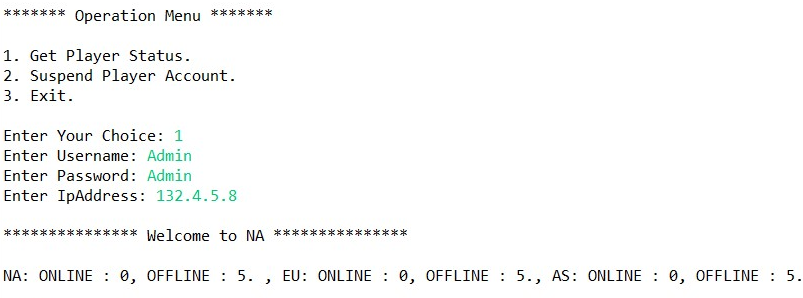
****

1. **Transfer Player Account (Successful – Player account exist and valid password and account with same username not exist on destination server.)**

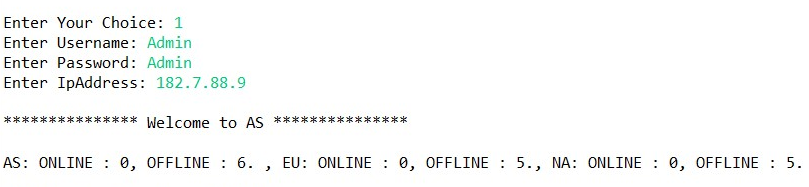
****

1. **Administrator Get Player Status.**

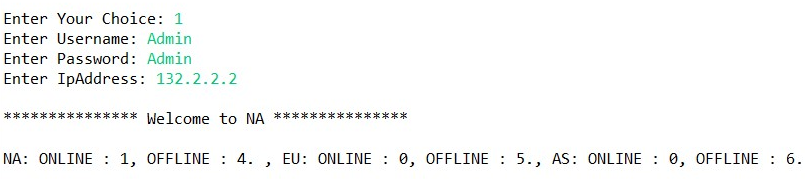
**Initially each server has 5 Player Accounts:**

****

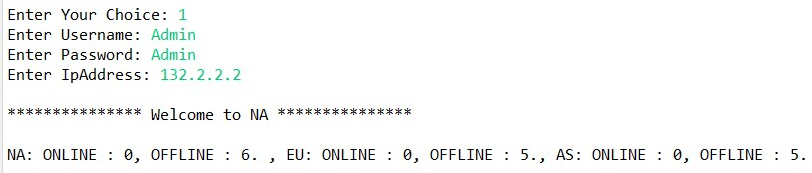
**After Creating One Player Account On Asia Server by playerClient:**



**After One Player Account Signed In at North-America Server by playerClient:**

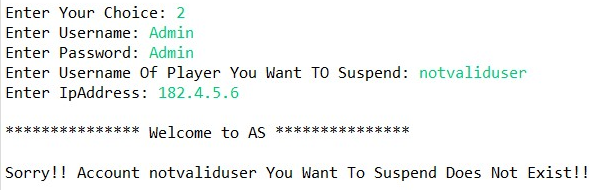


**After Transferring 1 Player Account from Asia Server to North-America Server.**

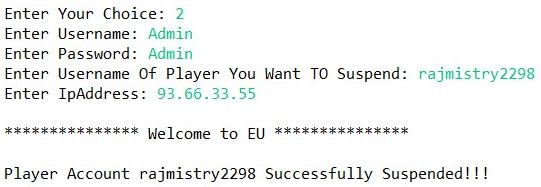


1. **Suspend Player Account.**

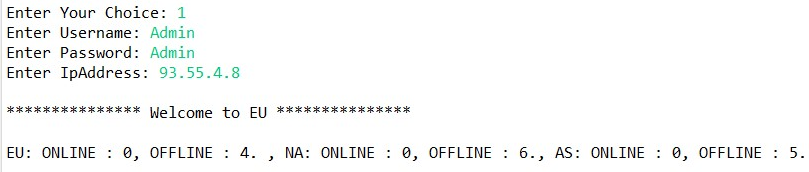
**Unsuccessful – Player Account with entered username not exist on server.**

****

**Suspend Player Account successfully.**

****

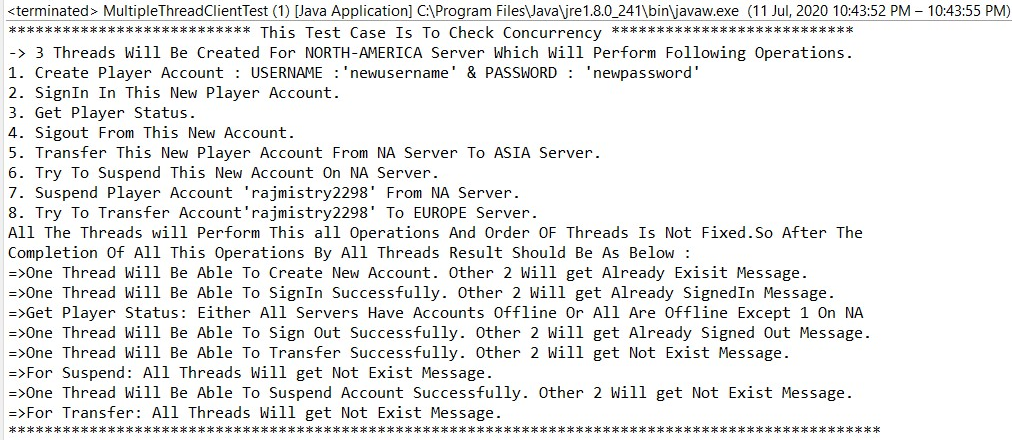
**Player Status after Suspending One Account from EU server.**

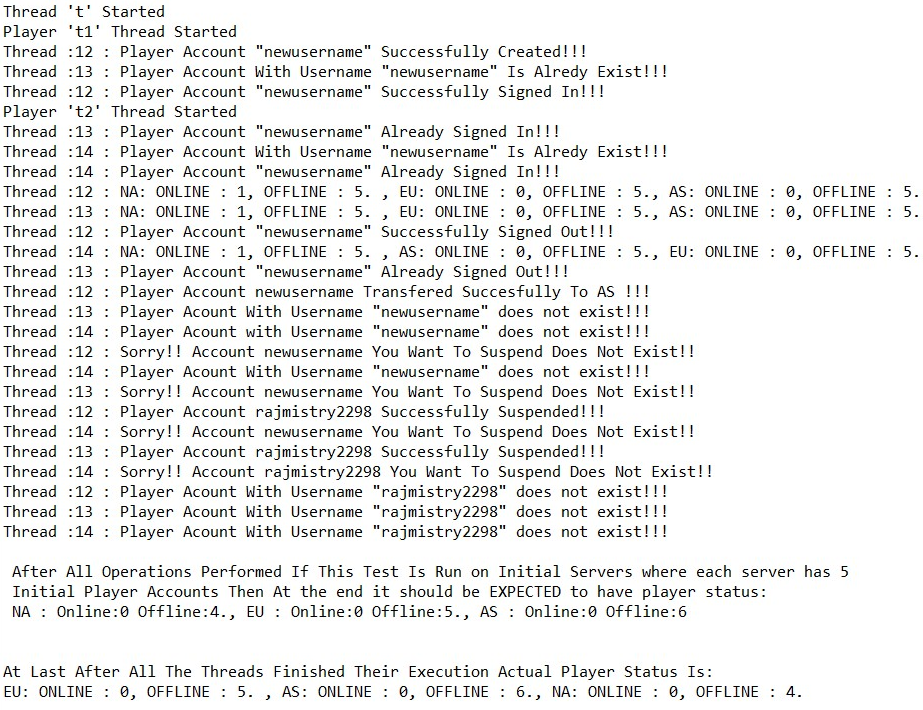
****

**Try to sign in into player account which was suspended by Admin.**

****

1. **Multithreaded Test For Checking Concurrency and Atomicity for Transfer Account And Suspend Account.**

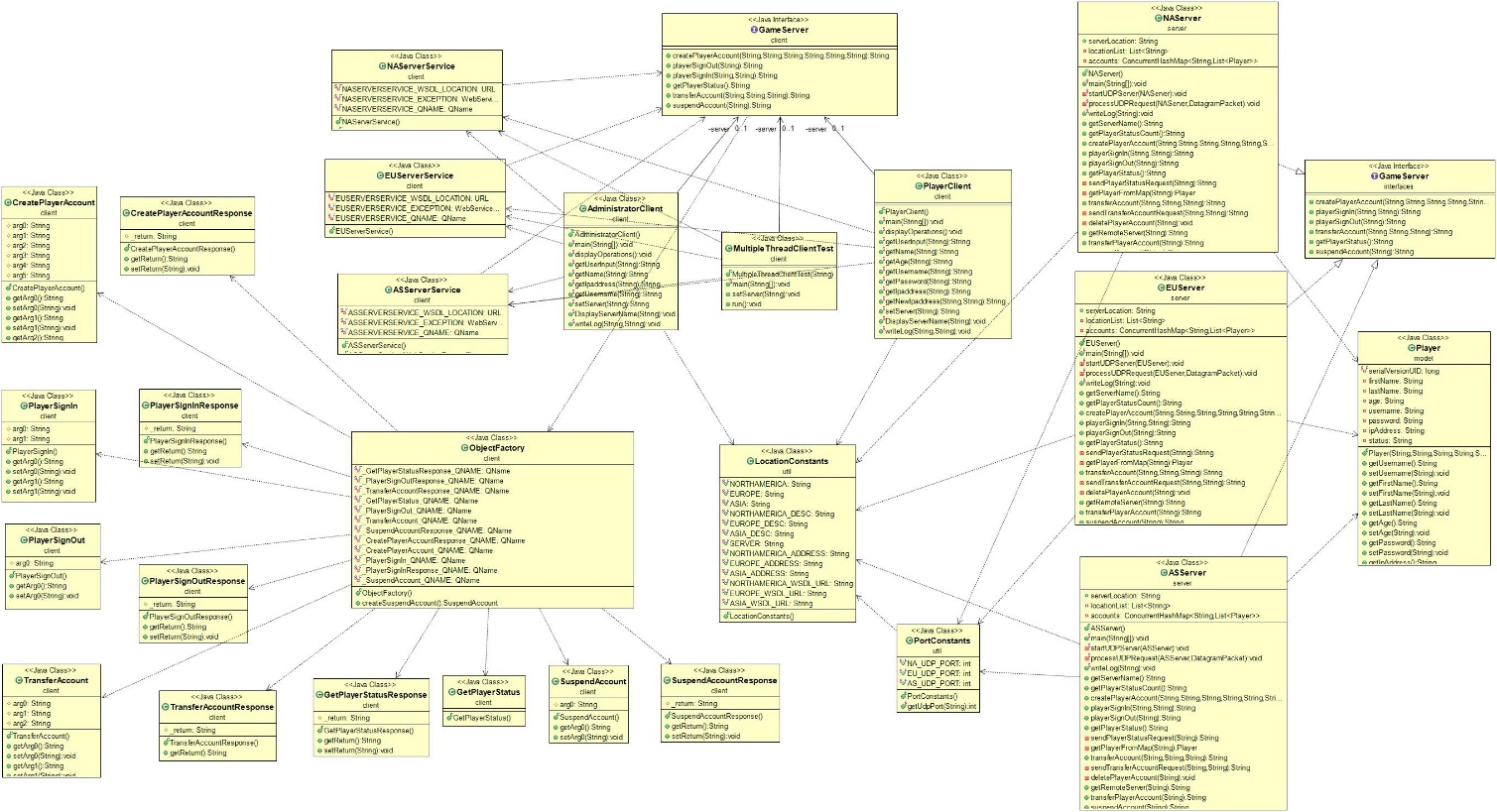
****

****

* **Challenges (Most Important / Difficult Part):**

1. Managing concurrency between 3 replicas.
2. FIFO Implementation & make UDP reliable.
3. Implementing synchronization among shared data and avoiding deadlocks.
4. Byzantine Error Implementation.

* **Class Diagram:**



* **NAServer, EUServer & ASServer:** These classes implement GameServer interface, start UDP server and create web services for North-America, Europe and Asia location respectively and publish the web services on below WSDL URLs,

<http://localhost:8080/NASever/NA?wsdl>

<http://localhost:8081/EUServer/EU?wsdl>

<http://localhost:8082/ASServer/AS?wsdl>

(We can see this in browser while servers are running)

* **GameServer.java (Service Endpoint Interface):** Thisis an interface which declares below methods that client can invoke on the service
* String createPlayerAccount (String firstName, String lastName, String age, String username, String password, String ipAddress);
* String playerSignIn (String username, String password);
* String playerSignOut (String username);
* String transferAccount (String username, String password, String newIpAddress);
* String getPlayerStatus ();
* String suspendAccount (String usernameToSuspend);
* **Player.java (Model):** This is a java bean class to hold following data of players,

First Name, Last Name, Age, Username, Password, Status and Ip Address.

* **NAServerService, EUServerService, ASServerService, GameServer, ObjectFactory, CreatePlayerAccount, CreatePlayerAccountResponse, PlayerSignIn, PlayerSignInResponse, PlayerSignOut, PlayerSignOutResponse, TransferAccount, TransferAccountResponse, GetPlayerStatus, GetPlayerStatusResponse, SuspendAccount & SuspendAccountResponse :** This All Java files are generated by wsgen and wsimport commands which are used in order to support webservices published by servers.
* **References:**
* https://www.tutorialspoint.com/webservices/what\_are\_web\_services.htm
* https://docs.oracle.com/cd/E19798-01/821-1841/gijti/index.html
* <http://bedford-computing.co.uk/learning/wp-content/uploads/2016/03/george-coulouris-distributed-systems-concepts-and-design-5th-edition.pdf>
* <https://www.infoworld.com/article/3215966/web-services-in-java-se-part-2-creating-soap-web-services.html>
* <https://www.infoworld.com/article/3215966/web-services-in-java-se-part-2-creating-soap-web-services.html?page=2>
* <https://www.youtube.com/watch?v=od6fNiegu-Q>
* <https://www.youtube.com/watch?v=-3w6LBl8E-8>