COMP 6231

Distributed System Design

Assignment 2 Report

Distributed Movie Ticket Management System (DMTMS)

Using Java IDL (CORBA –

Common Object Request Broker Architecture)

Submitted to: Professor Rajagopalan Jayakumar

By

Ishan Pansuriya: 40232841

TABLE OF CONTENTS

Overview

The Distributed Movie Ticket Booking System (DMTBS) is a distributed system where, a user which can be admin (Theater admin) or customer(user) perform operations like book, cancel, list the movie shows.

There are mainly 3 regions namely "Atwater (**ATW**)", "Verdun (**VER**)" and "Outremont (**OUT**)". User can be differentiated by their respective region's area code + their role (**A**-Admin, **C**-Customer) + 4-digit number (**0-9**).

Admin's Operations:

- Can add slot to the movie to their respective region.
- Can list movie shows available not only on his/her region but also to another region.
- Can remove movie slot only to their respective region.
- Can book movie in any of the 3 theaters.
- Can list booking schedule.
- Can cancel movie ticket.
- Can Exchange the movie tickets.

Customer's Operations:

- Can book movie tickets.
- Can list their own booked movie tickets.
- Can cancel their own booked movie tickets.
- Can exchange the movie tickets with new Movie Name.

There are 3 movies (Avenger, Avatar and Titanic) shows available in theaters. Each movie show has the movieId which includes area theater prefix (**ATW**/**VER**/**OUT**) + movie show timing (M-Morning, **E**-Evening, **A**-Afternoon) + date(**dd/mm/yyyy**) by which one movie show is identified.

UDP Ports: Atwater – 8001 / Verdun – 8002 / Outremont – 8003

Server: Atwater – "ATW" /Verdun – "VER" / Outremont – "OUT"

Architecture

1. As a data storage entity nested HashMap has been used in this assignment, which follows following structure.

```
HashMap<String, HashMap<String, Integer>> datastorage;
HashMap<String, HashMap<String, Integer>> user_data;
```

   - Outer key is used as a movie name (String).
   - Inside the value there is a another HashMap (<String, Integer>) which saves the movieId and booking capacity of theater respectively.

2. As a user data storage entity also a nested HashMap has been used which has the following structure.

   - Outer key is used to store customer-Id (String).
   - In value, another HashMap (<String, Integer>) has used, which includes movieName + movieId (Concatenated) as a key and bookedMovieTickets as a value.

3. CORBA (Common Object Request Broker Architecture) using Java IDL for Client/ Server (ATW, VER and OUT) interaction.

4. UDP to implement the communication between servers for fetching movie tickets available on other servers.

5. Multithreading used for parallel processing (I.e., multiple clients can send request similar time).

6. Logger is used to create logs of customer and admin separately not only that but also for each server there is a separate log file to log updates of every server.

7. Logs are stored in the location: src/Logs/

Starting the application:

1) Open command prompt
2) Cd into directory where .idl file is
3) Run the idl file “idlj -fall InterfaceOperation.idl”
4) Cd into newly created folder and compile the files “javac *.java”
5) Start the middleware “start orbd”
6) Run “java serverInstance”
7) Run “java client”

Class Diagram:

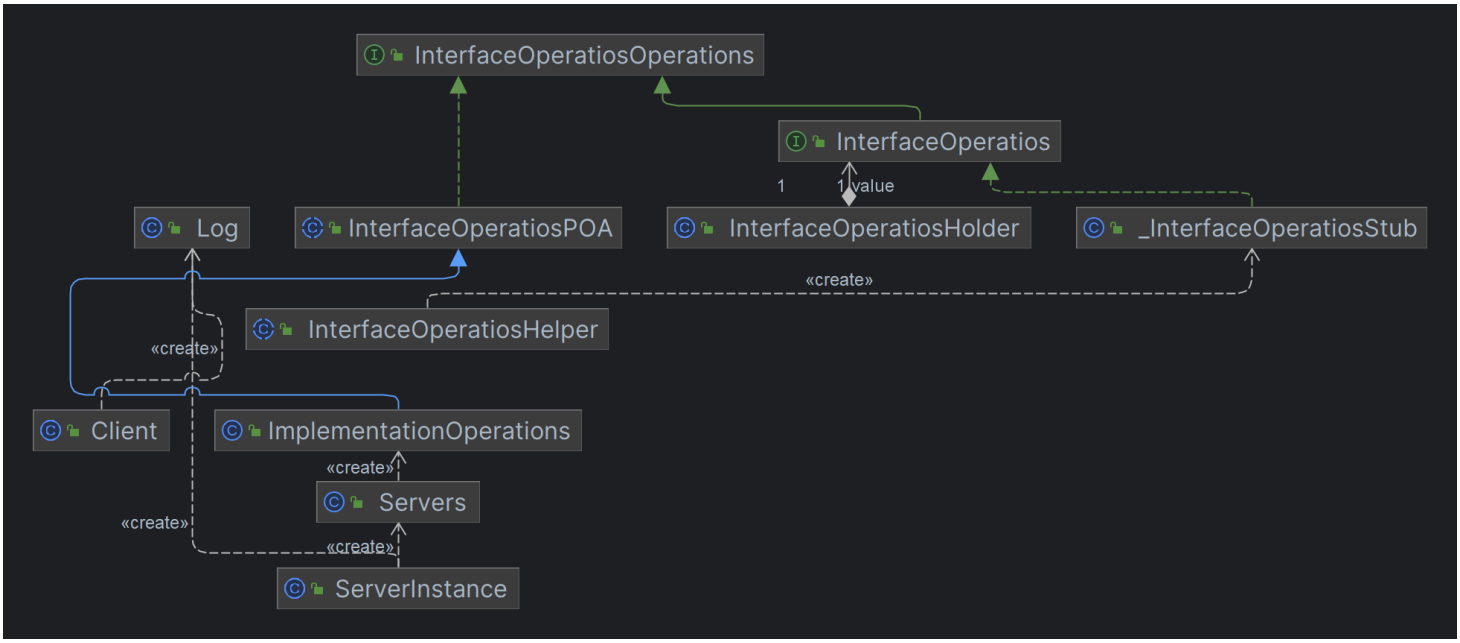
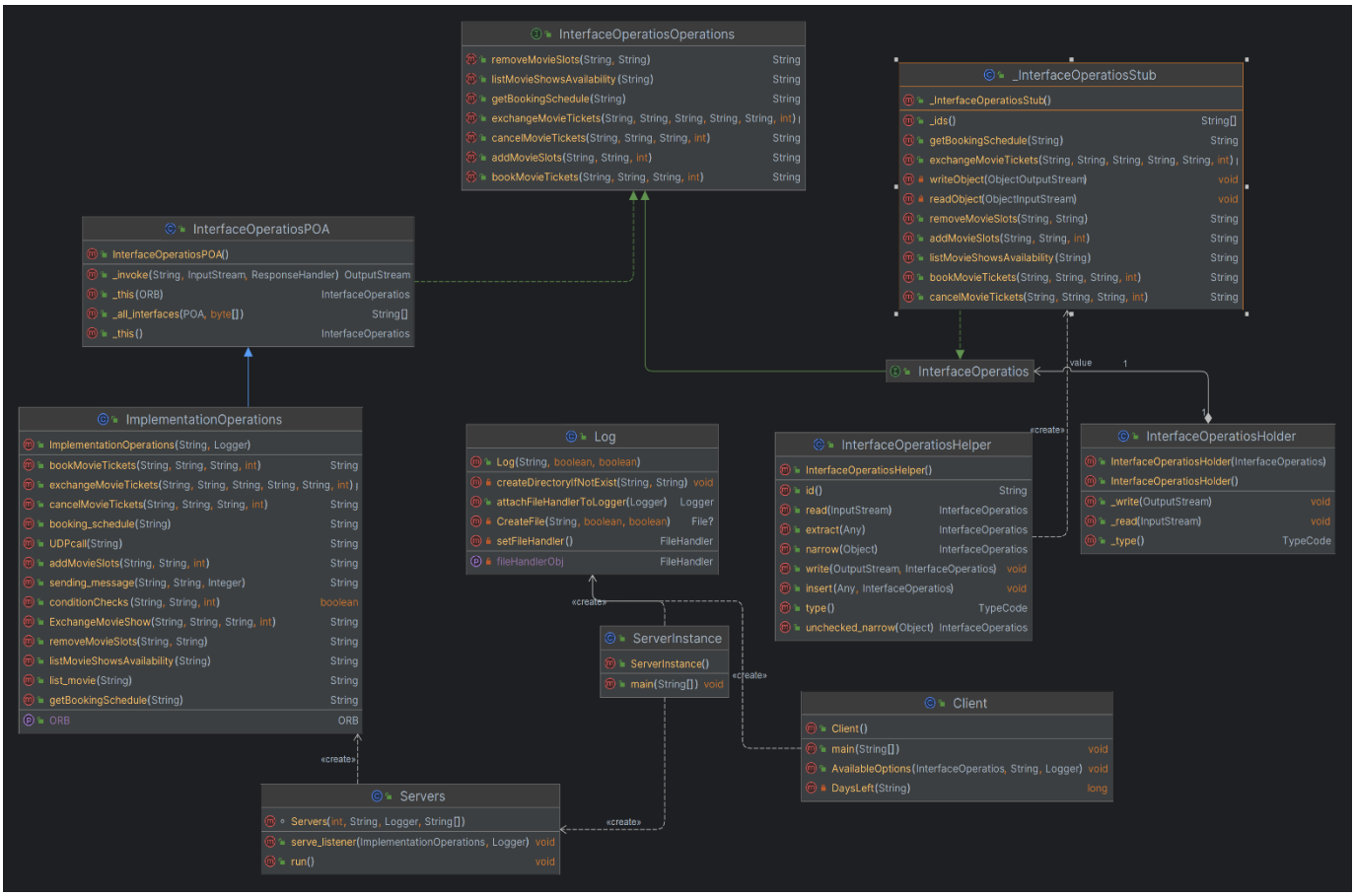
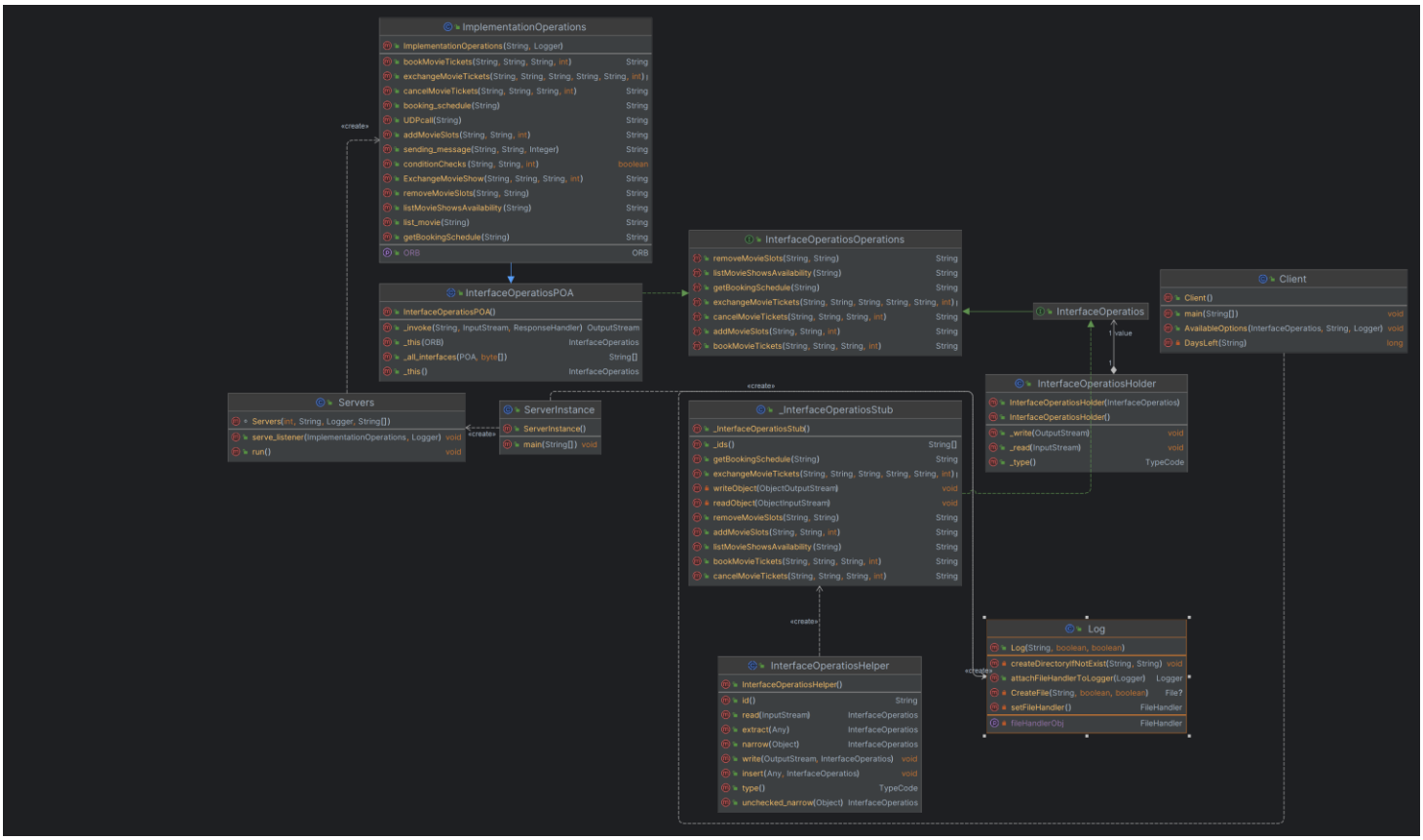


Diagram with Class Methods:



Implementation UML diagram:



Test Cases:

Name	Requirements	Scenario
Login	ID	• If prefix "A" in id's 4th Position.→ Can login as Admin • Else prefix "C" in id's 4th Position. →Can login as Customer
Admin	Add Slots	• Validate Admin ID • Check region by the prefix of adminId for unauthorized access. • Check the movieId is valid. • Check movieId is already in the HashMap then update capacity. • Can only able to add movie slots for a upcoming month. • Update logs client/server sides.

	Remove Slots	• Validate Admin Id • Check the region by prefix of the AdminId for unauthorized access. • Check the movieId provided is not of previous dates. • Check movieID is exist in HashMap. • Check user has not booked the tickets. • If user has booked the tickets then book them the next possible slot available. • In same theater, another movie shows with the same movieId should be untouched. • User no longer can book the movie show that is removed/being removed. • Update logs client/server sides.
	List Movies Shows	• Validate Admin Id • Check the region by prefix of the Admin for unauthorized access. • Check movie in own region. • UDP call for fetching data from the other region's server. • Show the data on display to user • Update the logs for server/client.
Customer/Admin	Book Movie	• Validate UserId for the booking tickets in their region. • Ask for the movie name/Check the movie name available in hashmap. • Ask for the MovieId & NumberOfTickets to book the movie in particular date/time/region. • User can book movie shows only 3 time/week outside of their regional server. • Unlimited booking in own regional server. • Update logs for server/client sides.

	Get booking Schedule	• Validate UserId. • Check if user has booked the tickets in their regional server. • UDP call to check if user has booked tickets in other regional server's • UDP call to fetch the data if booked tickets in other server. • Display the booking schedule. • Update the logs on server/client side.
	Cancel Booking	• Validate UserId with the booked ticket UserId. • Unauthorized if UserId is not same who booked the tickets. • Check whether booked ticket date is not passed the current date. • Delete the booked tickets and make it available in booking option. • Update the logs in serve/client side.
	Exchange MovieTickets	• Validate User with the booked ticket UserID. • Unauthorized if UserID not match. • Check whether booked ticket date is not passed the current date. • Check if provided new movie has the booking capacity to exchange or not. • Cancel the current movie Id. If fails then operation ends here. • Book the new movie with the same capacity. • Update the logs in the server/client side.
Logout	Option	Logout

Important part/ Difficulty

Important part of this assignment is interconnection communication of the server's using **UDP** without getting stuck in the infinite loop, also to run **CORBA** using Java IDL.

At first, I found while starting **Object Request Broker Daemon (ORBD)** facing the port-1050 connection refused error. Which was solved by changing the port to 1050 and setting the localhost properties.

I faced issue while running server instance, which was solved by running them on separate threads.