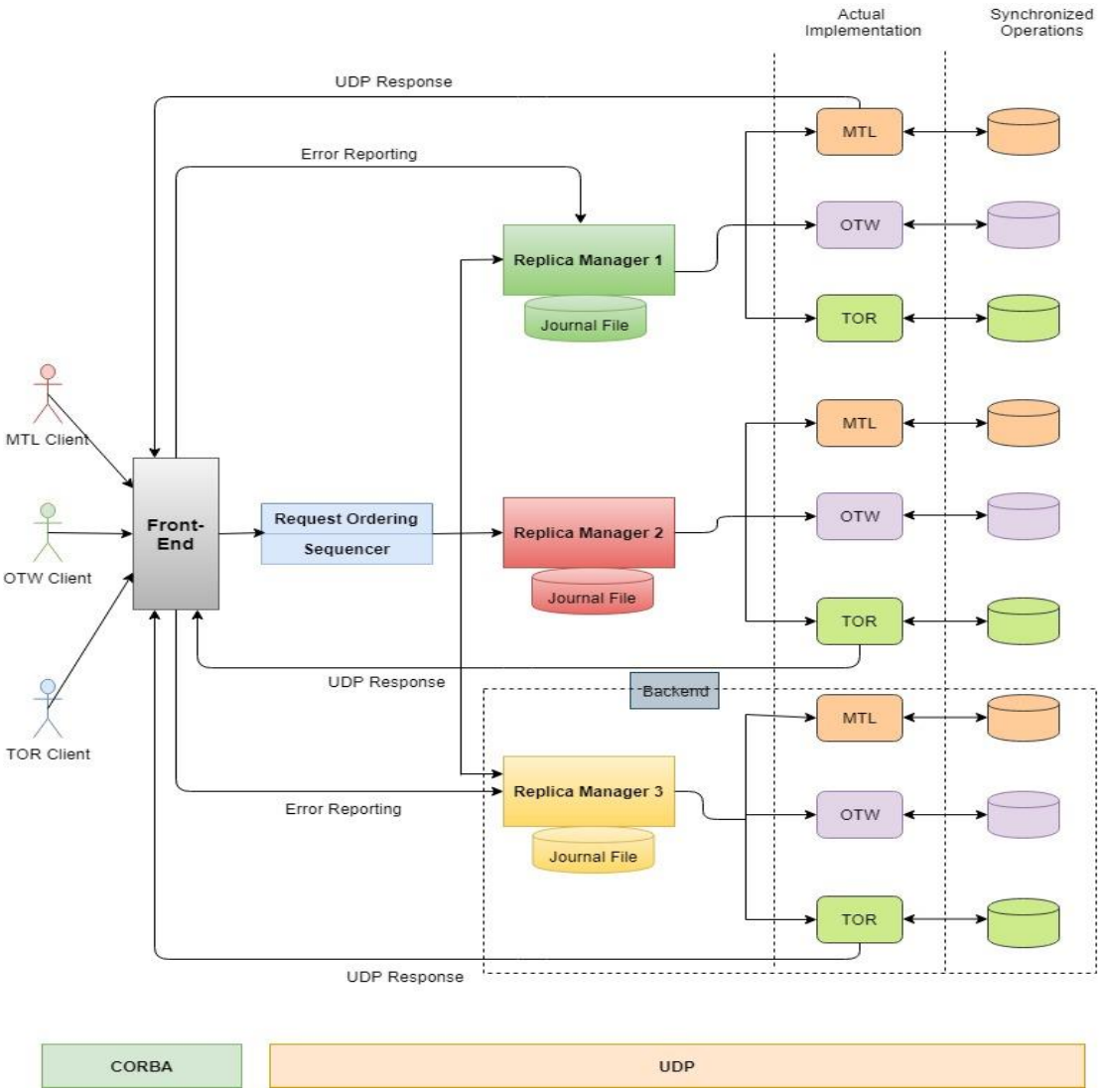
Distributed Event Management System

Prepared By: Himen Sidhpura (40091993), Jenny Mistry (40092281) & Ayush Dave(40080515)

Overall Description:

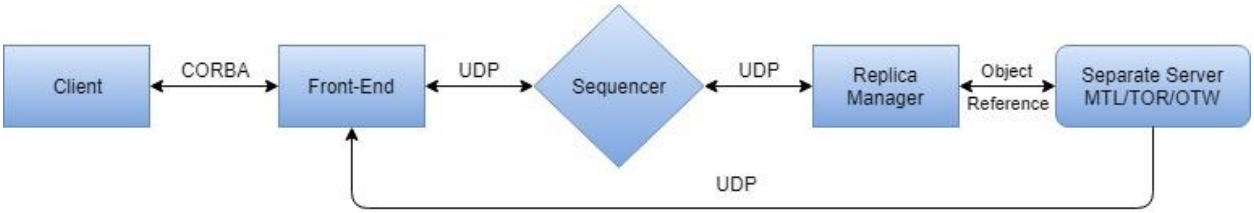
Event management is implemented as a distributed system to book and manage events across different branches of a corporate event management company. The system exposes CORBA Features and the users can see a single system handling user requests providing location and language transparency. It also manages simultaneous requests with adequate synchronization with the help of multithreading.

Design Architecture:

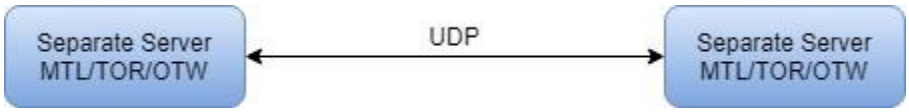


Communication about Components:

Communication paradigm among heterogeneous components



Communication paradigm among heterogeneous components



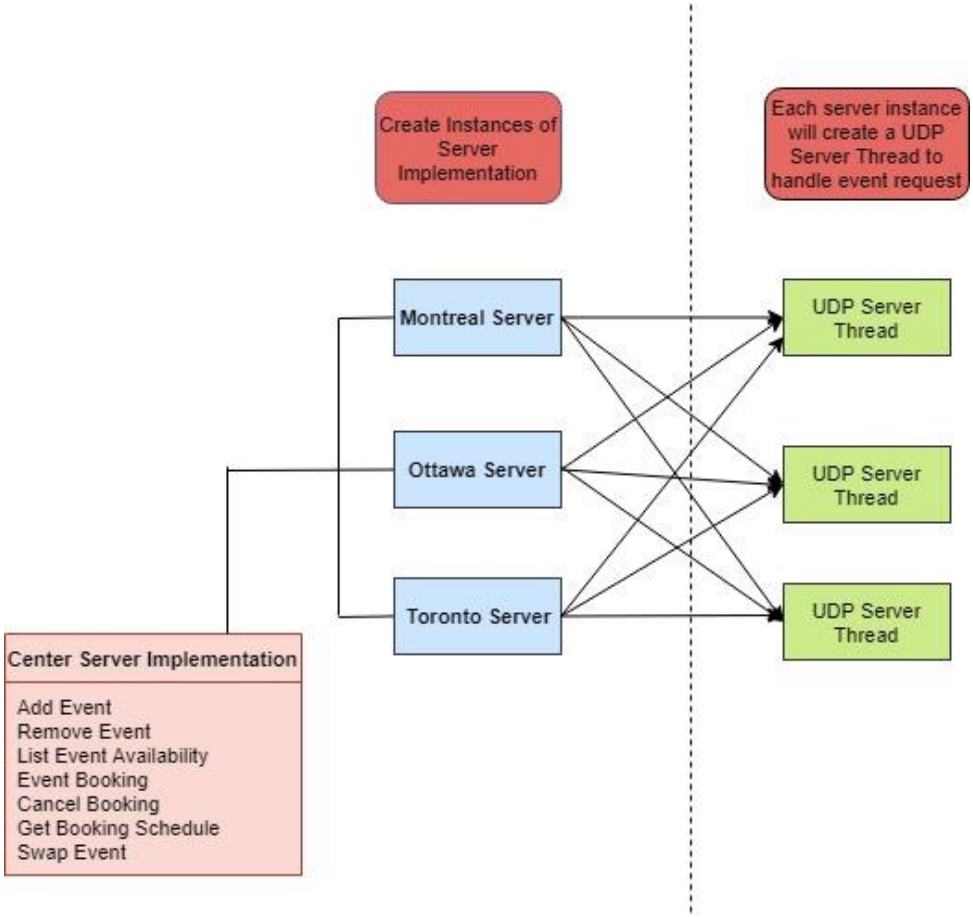
Overall Workflow:

- The client sends request to Front End using CORBA implementation.
- Front End register with CORBA Naming Service.
- After the FE forwards the request to sequencer, a unique sequence number is appended to the request and multicast to all the replica managers by the sequencer.
- Thus a sequencer keeps a track of order of all incoming requests.
- The replicas will store the request in a buffer and make sure that request is executed using total ordering. Replicas will buffer last processed results along with request id to avoid any duplicate request processing.
- The individual servers will process the request and send the result to FE.
- When the FE receives the response from the replicas, it perform the following:
    - The FE will inform about the error to the corresponding replica manager with the request id if any inconsistent result is found.
    - The FE will report a process crash to a replica manager if there is no response for a predefined amount of time from the replica.
    - The FE provides the response to the client after processing the responses from all the replicas.
- On receiving error reports thrice or completely no response from FE, the Replica Manager will announce the respective replica to be faulty and initiates the process of fault tolerance by restarting the server and resume the current state by synchronizing the data with the remaining server to maintain data consistency.

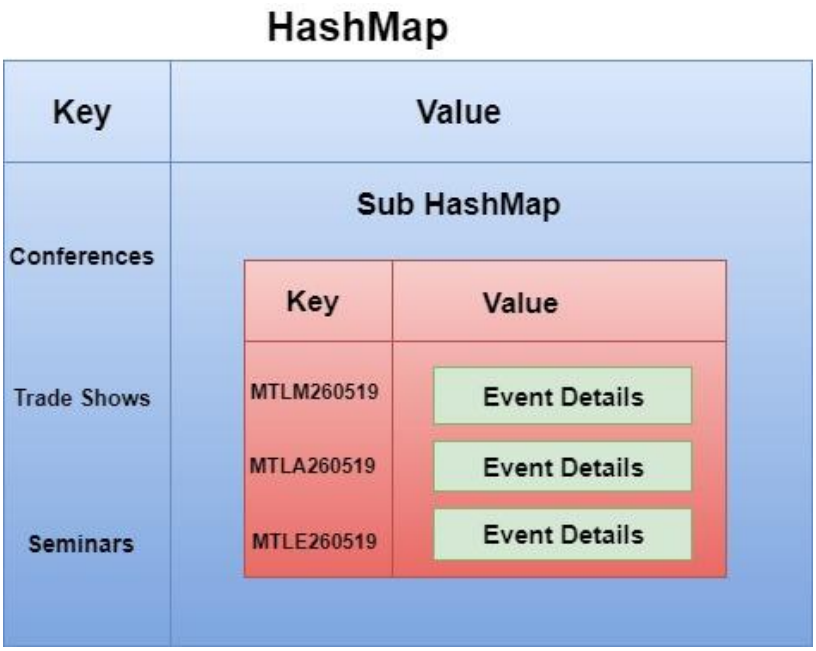
Intermediate Failure:

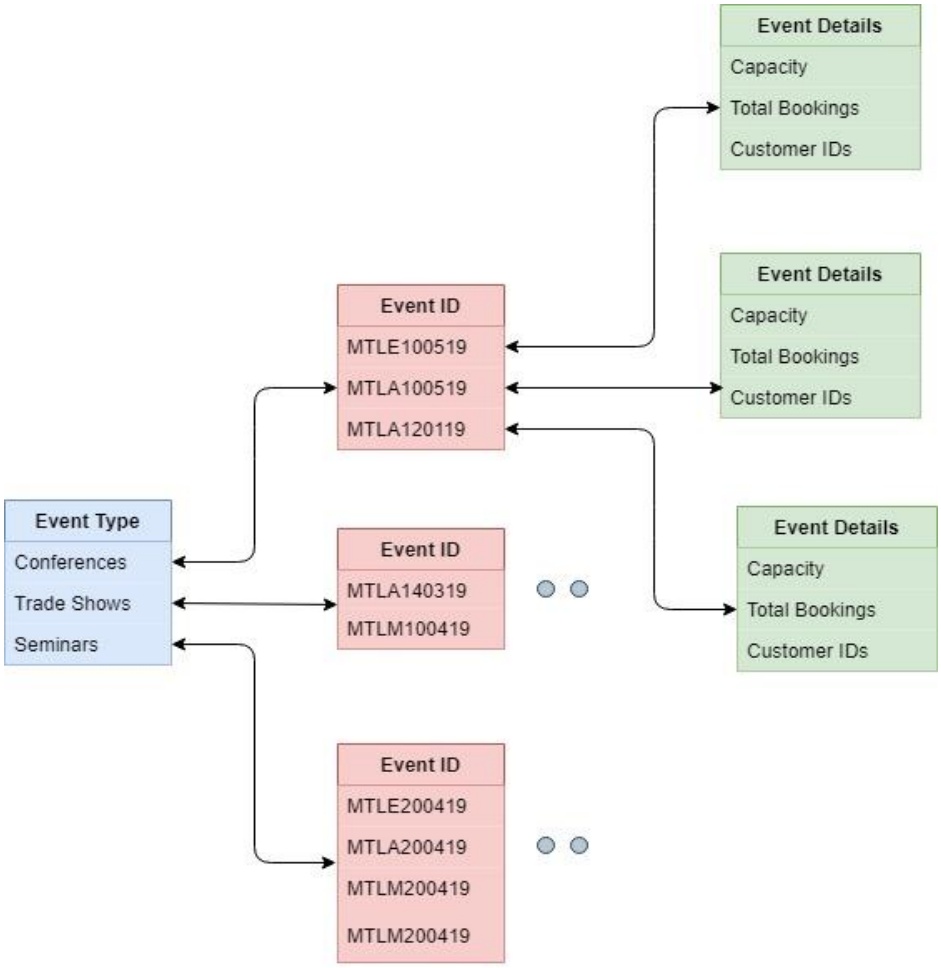
- On the event of third party attack, we have attempted to demonstrate the buggy behavior or corruption of messages in the system until three consecutive error reports from Front End.
- After the attacked replica is identified, failure recovery mechanism is initiated by the corresponding replica.
- The remaining replica will send their respective data to the affected replica to bring it back to the consistent state and make the data synchronized.

Server Architecture:



Data Models:


HashMap
Key
Value
Conferences
Trade Shows
Seminars
Sub HashMap
Key
Value
MTLM260519
Event Details
MTLA260519
Event Details
MTLE260519
Event Details


Event Details
Capacity
Total Bookings
Customer IDs
Event Details
Capacity
Total Bookings
Customer IDs
Event ID
MTLE100519
MTLA100519
MTLA120119
Event Type
Conferences
Trade Shows
Seminars
Event ID
MTLA140319
MTLM100419
Event Details
Capacity
Total Bookings
Customer IDs
Event ID
MTLE200419
MTLA200419
MTLM200419
MTLM200419

Logs:

To perform logging for troubleshooting on both server and client end, we have utilized the logger functionality of Java (java.util.logging).

Log Format:

Each log data comprises of the below mentioned details:

- Date and time the request was sent.
- Request type (book an event, cancel an event, etc.).
- Request parameters (clientID, eventID, etc.).
- Request successfully completed/failed.
- Server response for the particular request.

Center Server:

Each server log (Montreal, Ottawa, Toronto) will be saved in their respective folder

- logs/MTL.txt
- logs/OTW.txt
- logs/TOR.txt

These logs include:

- Event added
- Event cancelled
- Availability of events
- Events swapped

Client:

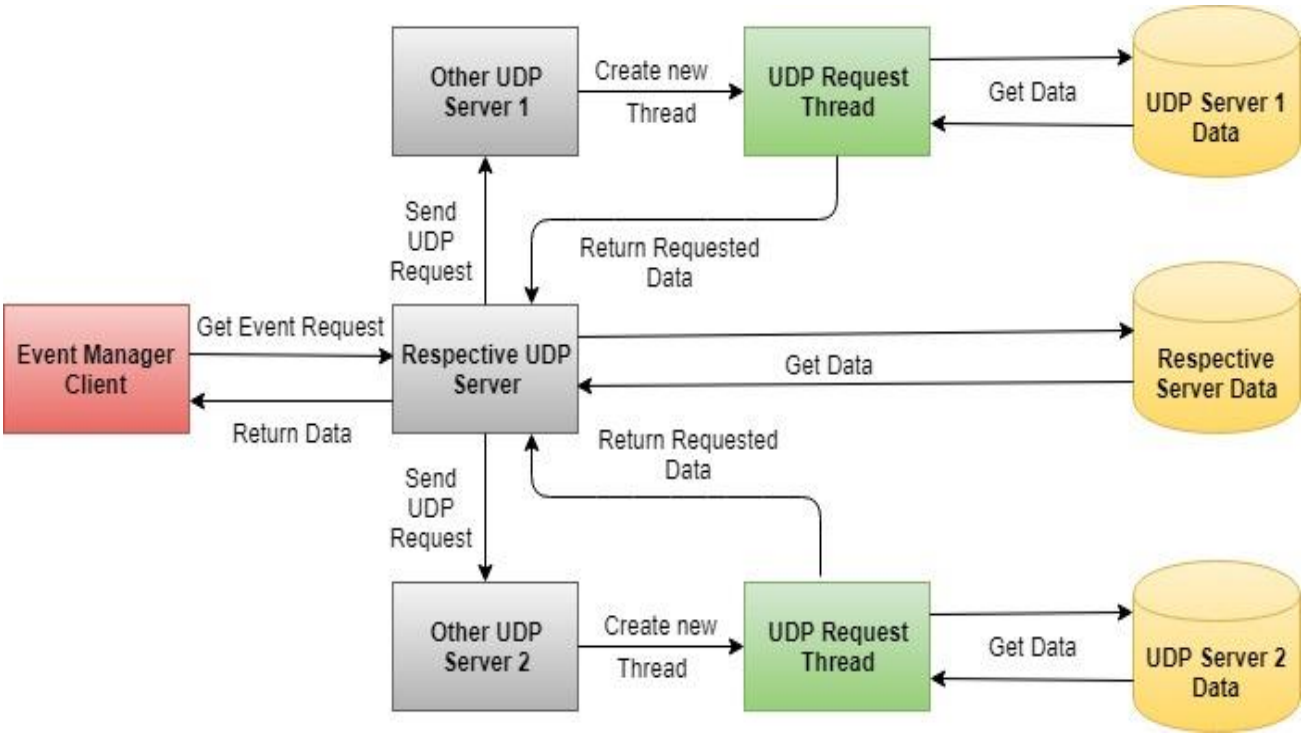
For every action performed by the client, a log file with clientID is created such as:

- Booking an event
- Canceling an event
- Retrieving booking schedule

Implementation:

- We have created a separate logger file for each of the three servers.
- To save contents of the corresponding log file, we have used a file handler.
- Various server responses are recorded using levels like WARNING, ERROR etc.

UDP Server Design:



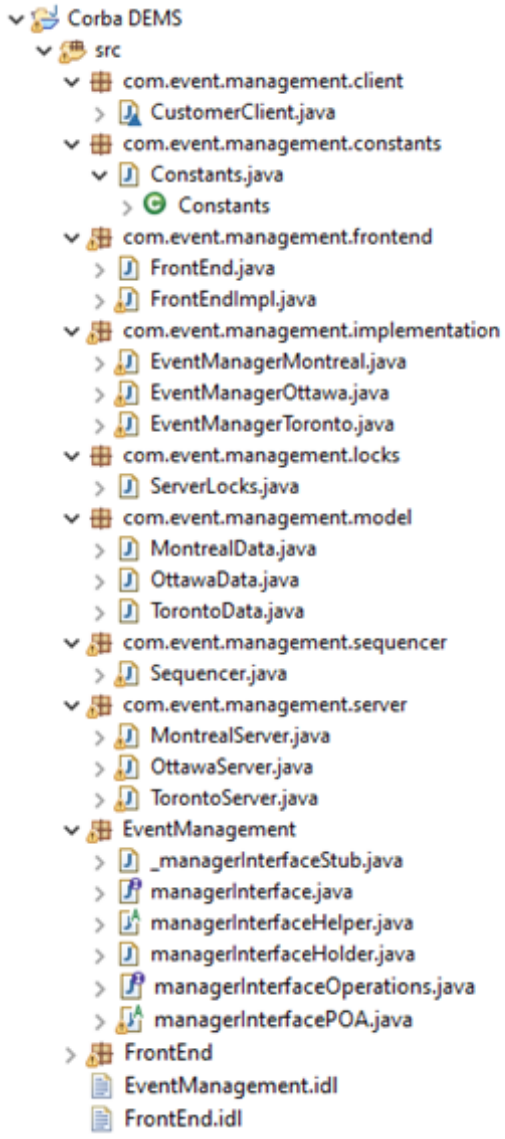
Flow:

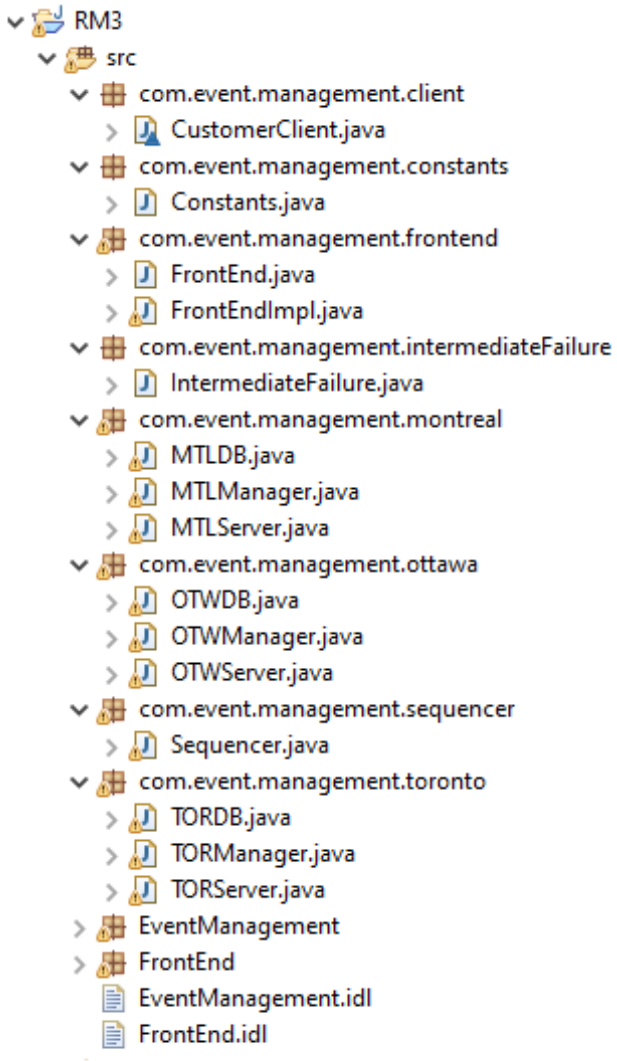
- The event manager client sends event request to the respective server.
- The server fetches the requested data.
- It forks new requests to send the event request to the other servers located at various locations.
- The UDP servers at these locations receives the request and creates new threads to process the request.
- The newly created threads fetches the respective data and responds to the request.
- The server which received the request responds to the manager client with appropriate data.

Concurrency:

The manager client creates new thread to communicate to each of servers to handle requests for same or different events at the same time.

Code Structure:





Challenges:

- Implementation of synchronization while managing multiple event requests at the same time has been challenging.
- The return types of various methods in CORBA Architecture were different for the replicas. Hence, we faced many challenges to change the return types of few to make the systems to coordinate with each other.

Test Scenarios:

- If the availability of an event is full, more customers cannot book the event.

- A customer can book as many events in his/her own city, but only at most 3 events from other cities overall in a month.
- A customer can perform only customer operation and cannot perform any event manager operation but an event manager can perform all operations for its own branches.
- If the user tries to add an event with an event id already added, then event details get updated.
- The user gets an error message “No events available”, if he/she tries to add an event which is not created by manager.
- All the user and manager event requests have been synchronized to handle multiple concurrent event requests for the same/different branches.
- The swap event is successful only if old event remove and new event add operation are successful.
- If old/new event does not exist for swap event, an error message is shown.
- The swap event throws an error if new event add operation exceeds the month’s max limit.
- After three consecutive error reports from a replica manager, the software bug is detected from the corresponding replica manager and is declared faulty.
- The system is made fault tolerant. Hence, when the fault is identified, the bug is resolved and the replica is up with the current server state.

References:

- https://www.geeksforgeeks.org/multithreading-in-java/
- https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html
- https://www.geeksforgeeks.org/client-server-software-development-introduction-to-common-object-request-broker-architecture-corba/
- https://www.math.uni-hamburg.de/doc/java/tutorial/idl/hello/idltojava.html
- https://xennis.org/wiki/CORBA_-_Advanced_example_with_server-client_in_Java_and_C%2B%2B
- https://docs.oracle.com/javase/7/docs/technotes/guides/idl/POA.html
- https://www.geeksforgeeks.org/synchronized-in-java/
- https://www.baeldung.com/java-broadcast-multicast
- https://www.geeksforgeeks.org/reentrant-lock-java/