CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231 - SUMMER 2019 Instructor and Coordinator: Sukhjinder K. Narula, R. Jayakumar

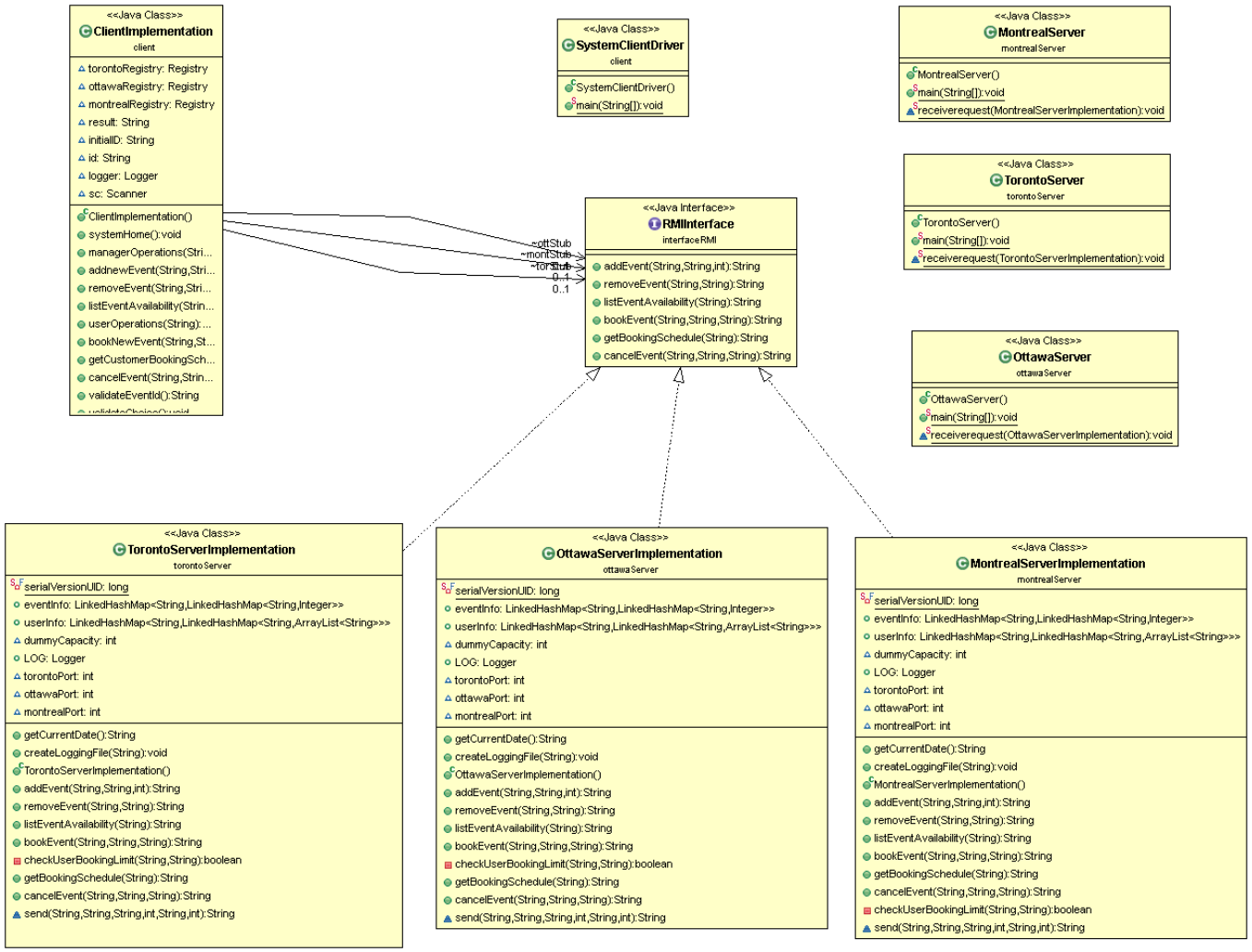
ASSIGNMENT 1

Student Id – 40093384 Name- Yogesh Nimbhorkar

Requirement: In this Assignment, we have been asked to implement a Distributed Event Management System with the help of the RMI (Remote Method Invocation) which was widely in use during past days.

ARCHITECTURE

Class Diagram



We have three servers: Montreal server, Toronto server and Ottawa server. I have designed an interface which consists of all the methods of customer and manager. This interface is then stored at client's end and server's end. Client use this information to know the method structure and definition. In turn, Server implements the methods of this interface.

CLIENT

I have structured the Customer Side in the client.pkg bundle which contains the SystemClientDriver class and ClientImplementation class. Client class founds facilitated registries for all the three servers for various systems i.e Montreal server, Toronto server and Ottawa server and likewise look for the binded objects of the servers from their separate registries to pronounce the stubs on the customer side thus characterizing the way to make RPC utilizing Java RMI.

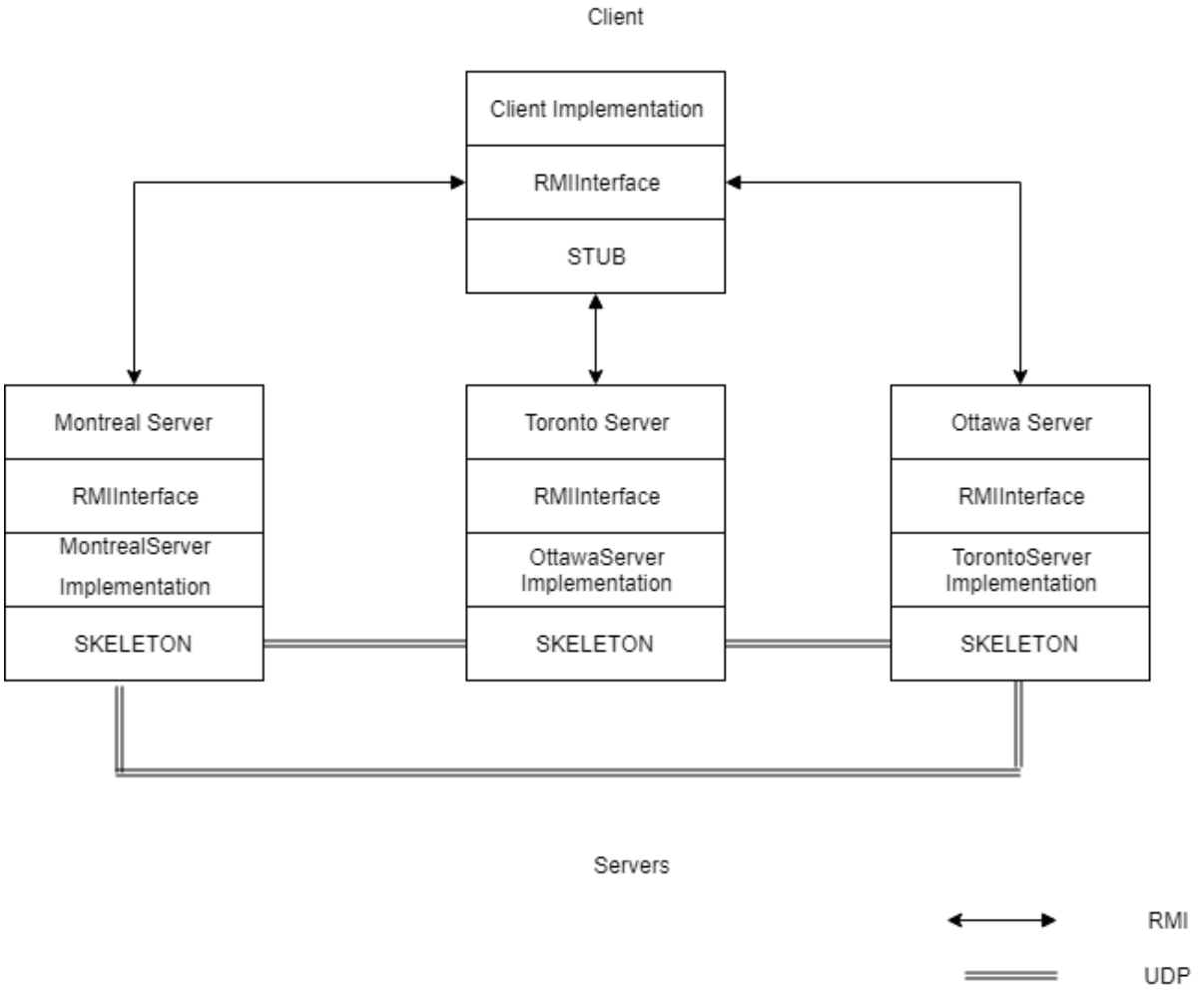
SERVER

In this case, I have designed six server classes for all the three servers respectively.

- MontrealServer.java
- MontrealServerImplementation.java
- OttawaServer.java
- OttawaServerImplementation.java
- TorontoServer.java
- TorontoServerImplementation.java

All the three server classes implement an interface RMIInterface.java and the implementation classes for all the three servers as MontrealServerImplementation.java , OttawaServerImplementation.java and TorontoServerImplementation.java and the functionality is common for all the three servers as they all have similar capabilities. I have created registry for each of the three server on different ports and binded their objects i.e skeleton with the references on that registry. These servers can also communicate with each other using UDP and exchange messages.

Below is the architectural diagram of my application –



This diagram illustrates simple client server architecture using java RMI technology. Below are the main points that focuses on this architecture –

1. Client and Server communicate with each other using Java RMI. Client invokes remote methods implemented in the server using stubs.
2. There is a common interface which has the definition of all the user and manager methods. All the three servers shown above i.e Montreal, Ottawa and Toronto system servers implement the methods of this server.
3. Servers can communicate with each other using only UDP.

There are three methods of user and manager each that have been implemented. The significance of each method is mentioned below –

Manager Role:-

Add Event –
When an event manager invokes this method through the server associated with this event manager (determined by the unique eventManagerID prefix), attempts to add an event with the information passed, and inserts the record at the appropriate location in the hash map. The server returns information to the event manager whether the operation was successful or not and both the server and the client store this information in their logs. If an event already exists for same event type, the event manager can't add it again for the same event type but the new bookingCapacity is updated. If an event does not exist in the database for that event type, then add it.

Remove Event –
When invoked by an event manager, the server associated with that event manager (determined by the unique eventManagerID) searches in the hash map to find and delete the event for the indicated eventType and eventID. Upon success or failure it returns a message to the event manager and the logs are updated with this information. If an event does not exist, then obviously there is no deletion performed. Just in case that, if an event exists and a client has booked that event, then, delete the event and take the necessary actions.

List Event Availability –
When an event manager invokes this method from his/her branch's city through the associated server, that branch's city server concurrently finds out the number of spaces available for each event in all the servers, for only the given eventType. This requires inter server communication that will be done using UDP/IP sockets and result will be returned to the client. Eg: Seminars - MTLE130519 3, OTWA060519 6, TORM180519 0, MTLE190519 2.

User Role:-

1. **Book Event** –
   When a customer invokes this method from his/her city through the server associated with this customer (determined by the unique customerID prefix) attempts to book the event for the customer and change the capacity left in that event.
2. **Get Schedule** –
   When a customer invokes this method from his/her city's branch through the server associated with this customer, that city's branch server gets all the events booked by the customer and display them on the console.
3. **Cancel Event** –
   When a customer invokes this method from his/her city's branch through the server associated with this customer (determined by the unique customerID prefix) searches the hash map to find the eventID and remove the event.

Data Structures : -

1. **Evnet Information HashMap** – This is a nested hashmap which stores the information of an item in the system. It contains key as Event Type and its value as a hashmap which in turn stores Event Id as a key and capacity as the value.

   ```
   public HashMap<String, HashMap<String, Integer>> eventInfo = new HashMap<String, HashMap<String, Integer>>>();
   ```

2. **User Information HashMap** – This map contains user ID as the key and sub hashmap of Event Type and array list of event ids as values.

   ```
   public HashMap<String, HashMap<String, ArrayList<String>> userInfo = new public HashMap<String, HashMap<String,, ArrayList<String>>>();
   ```

TEST SCENARIOS:

ADD Operation

1. A manager adds an event in the system. If it is already present, the capacity is updated otherwise a new event is added in the system.
2. A manager can add event of other system.
3. A manager cannot enter a negative capacity of an event while adding it in the system.
4. A manager can book, cancel or check the schedule on behalf of any user.

REMOVE Operation

5. Manager can only remove an event If it already exists otherwise an error message will be returned from the server, event will be automatically removed from the user's list who have booked the event.
6. Manager can update the capacity of event.

CHECK AVAILABILITY Operation

7. All the events will be returned to the manager with respect to event type. Event id and capacity of event.

BOOK EVENT Operation

8. User as well as manager can book the event of same or different server.
9. Check for event is not full.
10. Check if event is present or not.

CHECK SCHEDULE Operation

11. User / manager can check the schedule of the particular userid.

CANCEL EVENT Operation

12. A user can cancel event or manager on behalf of user. Provided the event type and event id is present in the system.

Important Points from this Assignment:-

1. The applications has been designed in such a way that a user cannot access any operations which are meant for the manager. Similarly, a manager can access operations of user.
2. All validations have been performed at the client side to make sure that only those with valid user Id and manger Id are allowed access to the application. Also, invalid event Id's are not allowed.
3. Manager can delete the event from the system and all the users will be deregistered from the event.
4. The methods of the server are synchronized so that multiple users cannot access the same method at a time.
5. Mutiple servers have the ability to communicate messages with each other using UDP protocol thus enabling server to server communication possible.

Difficult Points from this Assignment:-

1. Using nested Hash map to store items information in a server was little challenging.
2. Implementing logging in this application as I was confused whether to use Log4j or java logging framework and which one is more efficient. Creating a log file for every user required more depth understanding of file handling.
3. Initiating communication between different servers using UDP.
4. Most important, designing the architecture and flow of the application.

References:-

1. Tutorial Point
2. JavaTPoint
3. GeeksforGeeks
4. Oracle docs