# COMP6231 Assignment 1

**Tianlin Yang 40010303**
**Gaoshuo Cui 40085020**

A Report
In the Department
of
Computer Science & Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Assignment 1 of

COMP6231 (Distributed System) at

Concordia University

Montreal, Quebec, Canada

June 2019

# TABLE OF CONTENTS

# 1 Overview

The Distributed Event Management System (DEMS) is a distributed system for a leading corporate event management company: a distributed system used by an event manager who manages the information about the events and customers who can book or cancel an event across the company's different branches. There have three branches in different cities: Toronto (TOR), Montreal (MTL) and Ottawa (OTW) for our implementation.

The users of the system are event managers and event booking customers. Event Managers and customers are identified by a unique manager ID and customer ID respectively. (e.g.: MTLC1234, OTWM1234). A customer can book, drop events and view his schedule whereas a Manager perform all the student operations and even can add/drop courses.

Each server maintains its internal in-memory database, basically composed of a HashMap, which it uses to store various information and a user interacts with the server using the Java RMI connection. The inter server communication is done using UDP sockets. Each server maintains a log file for all the operation performed on it. A log file per user login is also maintained.

To the make the system more robust, after successful login, the user interaction is performed on a separate thread. Each logged in user communicates with its corresponding department server and performs necessary operations. Since multiple users access a server concurrently, so the proper synchronization of data is implemented in the code for thread safe communication. Moreover, the user input is case insensitive.

## 1.1 Develop tools

All code writing in Java IDE Eclipse, Java IDK version is Java 8. In order to generate Diagram for classes in the project, Object Aid Plugin has been used.

# 2 Functional representation

## 2.1 Running the system

For UI:

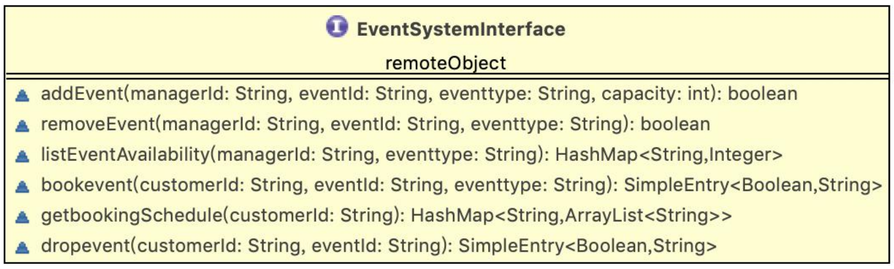
1) Start the OTW_Server, TOR_Server and then MTL_Server.
2) Start the Client program ‘client’.

For multithreads test:

1) Start the OTW_Server, TOR_Server and then MTL_Server.
2) Start the MultithreadsTest ‘Datainitialize’ and ‘MultiClient’.

## 2.2 Writing steps of the system

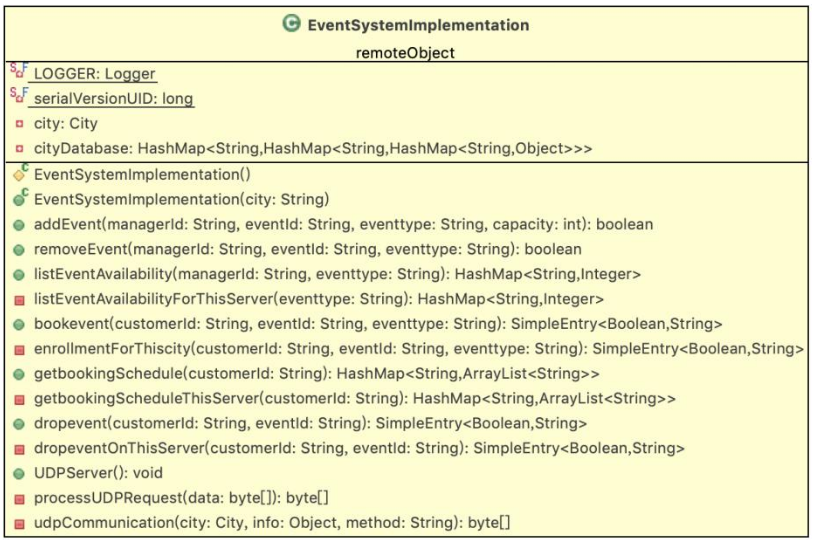
A. Define the remote interface



EventSystemInterface
remoteObject

- addEvent(managerId: String, eventId: String, eventtype: String, capacity: int): boolean
- removeEvent(managerId: String, eventId: String, eventtype: String): boolean
- listEventAvailability(managerId: String, eventtype: String): HashMap<String,Integer>
- bookevent(customerId: String, eventId: String, eventtype: String): SimpleEntry<Boolean,String>
- getbookingSchedule(customerId: String): HashMap<String,ArrayList<String>>
- dropevent(customerId: String, eventId: String): SimpleEntry<Boolean,String>

B. Develop the implementation class

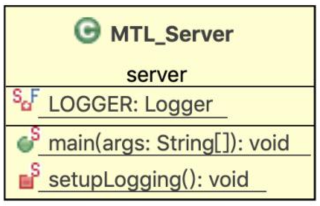


EventSystemImplementation
remoteObject

- LOGGER: Logger
- serialVersionUID: long
- city: City
- cityDatabase: HashMap<String,HashMap<String,HashMap<String,Object>>>

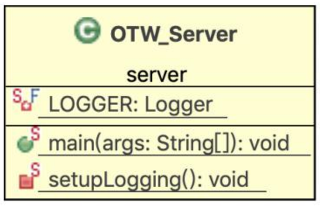
- EventSystemImplementation()
- EventSystemImplementation(city: String)
- addEvent(managerId: String, eventId: String, eventtype: String, capacity: int): boolean
- removeEvent(managerId: String, eventId: String, eventtype: String): boolean
- listEventAvailability(managerId: String, eventtype: String): HashMap<String,Integer>
- listEventAvailabilityForThisServer(eventtype: String): HashMap<String,Integer>
- bookevent(customerId: String, eventId: String, eventtype: String): SimpleEntry<Boolean,String>
- enrollmentForThiscity(customerId: String, eventId: String, eventtype: String): SimpleEntry<Boolean,String>
- getbookingSchedule(customerId: String): HashMap<String,ArrayList<String>>
- getbookingScheduleThisServer(customerId: String): HashMap<String,ArrayList<String>>
- dropevent(customerId: String, eventId: String): SimpleEntry<Boolean,String>
- dropeventOnThisServer(customerId: String, eventId: String): SimpleEntry<Boolean,String>
- UDPServer(): void
- processUDPRequest(data: byte[]): byte[]
- udpCommunication(city: City, info: Object, method: String): byte[]

C. Develop the server program



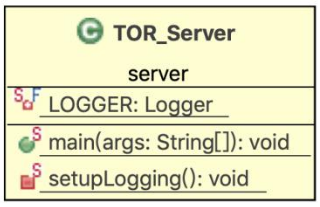
MTL_Server
server

- LOGGER: Logger
- main(args: String[]): void
- setupLogging(): void



OTW_Server
server

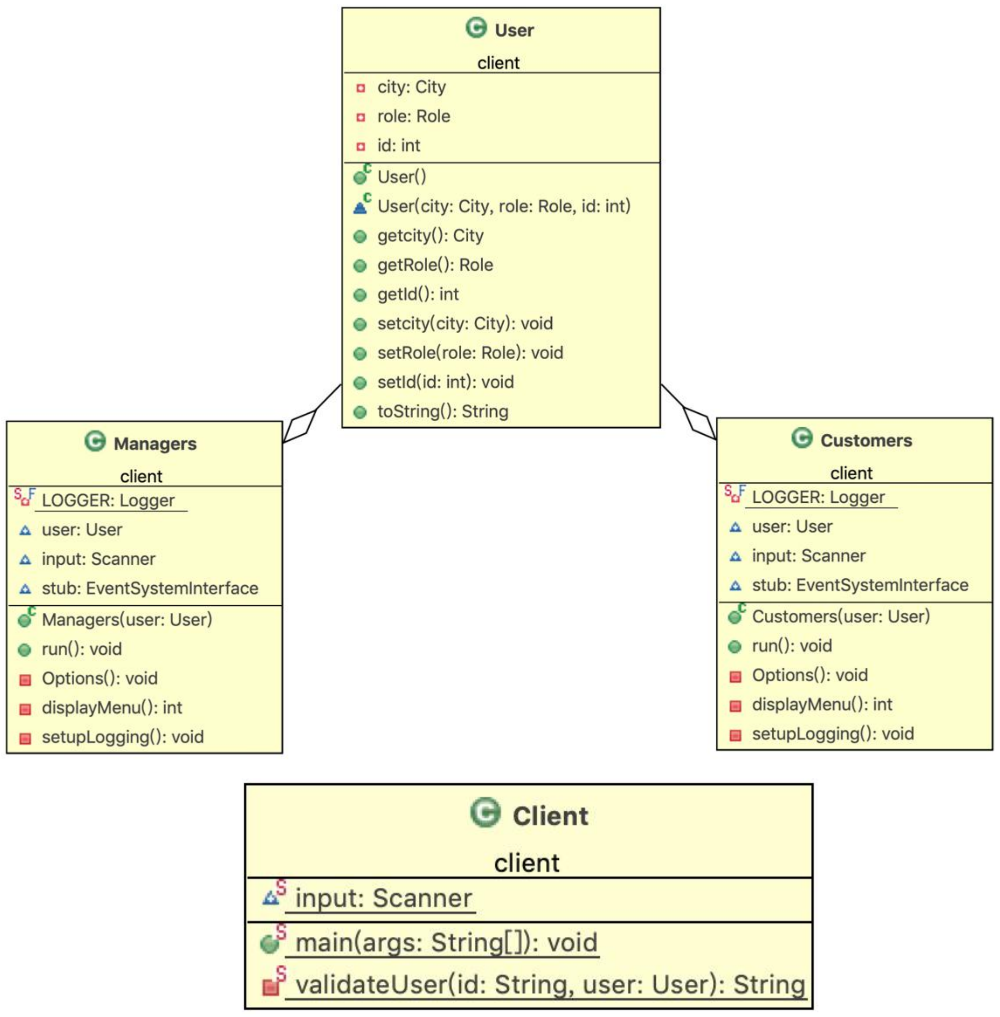
- LOGGER: Logger
- main(args: String[]): void
- setupLogging(): void



TOR_Server
server

- LOGGER: Logger
- main(args: String[]): void
- setupLogging(): void

D. Develop the client program



E. Compile the application

F. Execute the application

# 3 Architecture of system

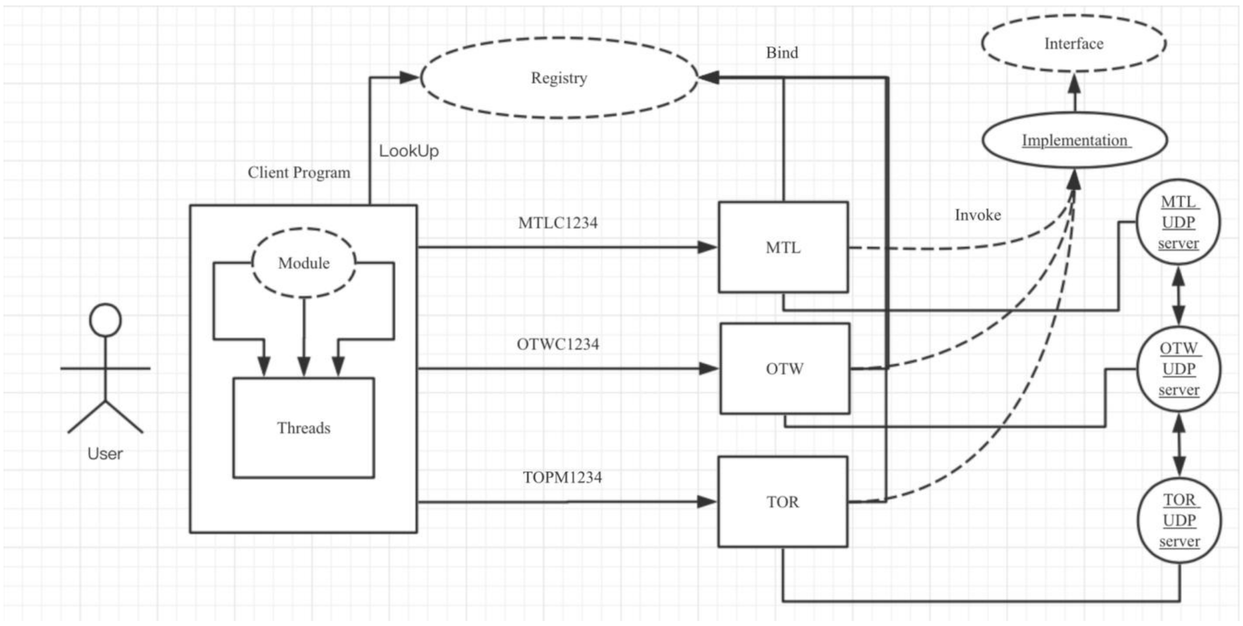
Three different servers (MTL, OTW, TOR) are started, which then start their own UDP servers for socket communication. Each of these servers bind their object reference in the registry using a unique name.

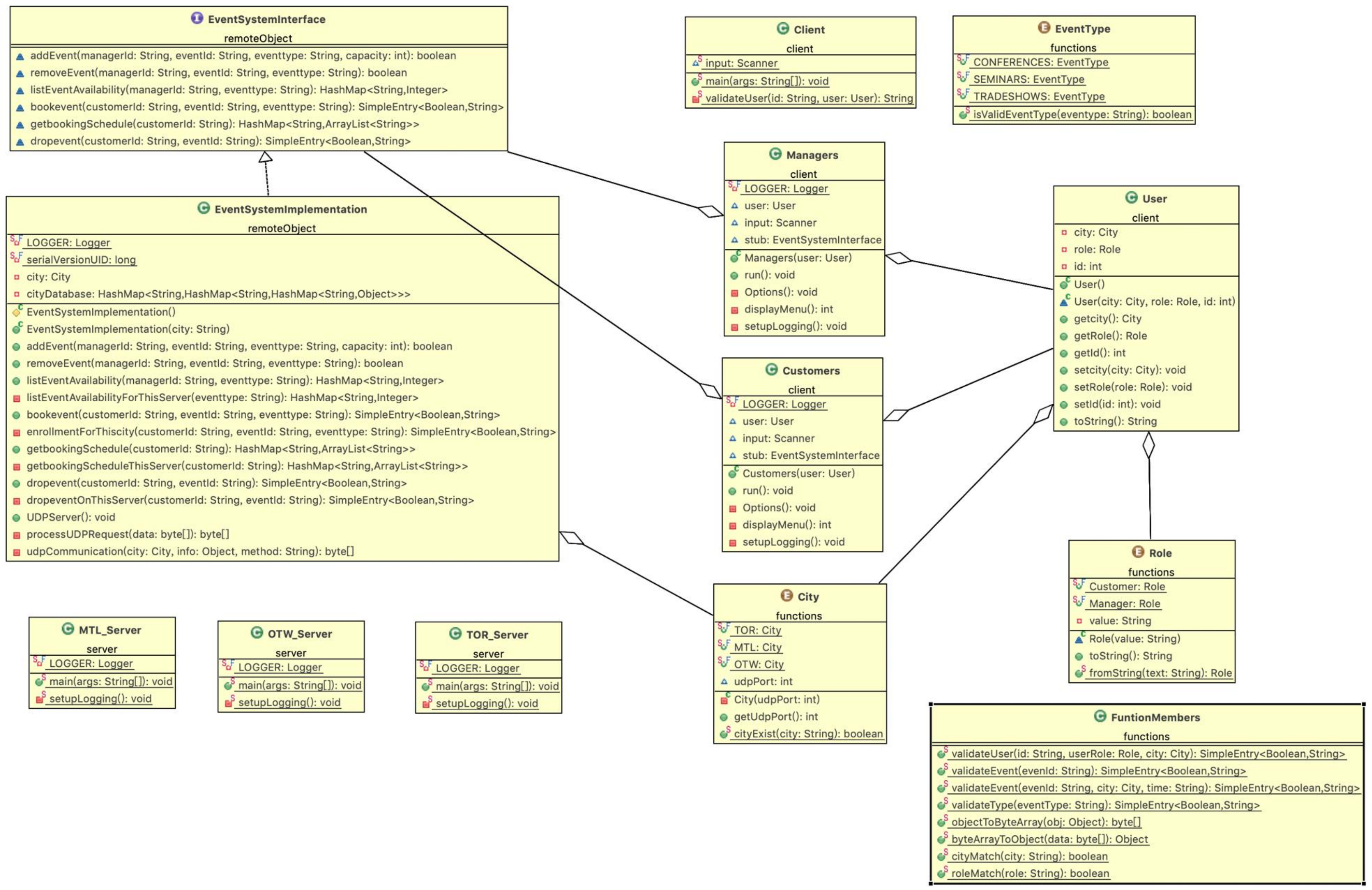
Apart from these servers, there's a client program which handles the user interaction on the client program, a user log-in using its unique ID. This unique ID is validated to identify the server locations, role and ID of the user.

After successful login, a separate thread is started to handle the requests from this user. Since a thread is a light weight process, so this make the system more robust and responsive.

In the thread, we consult the registry to get reference of its corresponding server. Then after, based on the user type (manager or customer), a list of available operations is displayed.

In particular, a user only communicates with its corresponding server. But if the operation requires data from other departmental servers, then the user server makes a UDP request to the target server and gets the required data using a socket communication.




EventSystemInterface
remoteObject
addEvent(managerId: String, eventId: String, eventtype: String, capacity: int): boolean
removeEvent(managerId: String, eventId: String, eventtype: String): boolean
listEventAvailability(managerId: String, eventtype: String): HashMap<String,Integer>
bookevent(customerId: String, eventId: String, eventtype: String): SimpleEntry<Boolean,String>
getbookingSchedule(customerId: String): HashMap<String,ArrayList<String>>
dropevent(customerId: String, eventId: String): SimpleEntry<Boolean,String>
EventSystemImplementation
remoteObject
LOGGER: Logger
serialVersionUID: long
city: City
cityDatabase: HashMap<String,HashMap<String,HashMap<String,Object>>>
EventSystemImplementation()
EventSystemImplementation(city: String)
addEvent(managerId: String, eventId: String, eventtype: String, capacity: int): boolean
removeEvent(managerId: String, eventId: String, eventtype: String): boolean
listEventAvailability(managerId: String, eventtype: String): HashMap<String,Integer>
listEventAvailabilityForThisServer(eventtype: String): HashMap<String,Integer>
bookevent(customerId: String, eventId: String, eventtype: String): SimpleEntry<Boolean,String>
enrollmentForThiscity(customerId: String, eventId: String, eventtype: String): SimpleEntry<Boolean,String>
getbookingSchedule(customerId: String): HashMap<String,ArrayList<String>>
getbookingScheduleThisServer(customerId: String): HashMap<String,ArrayList<String>>
dropevent(customerId: String, eventId: String): SimpleEntry<Boolean,String>
dropeventOnThisServer(customerId: String, eventId: String): SimpleEntry<Boolean,String>
UDPServer(): void
processUDPRequest(data: byte[]): byte[]
udpCommunication(city: City, info: Object, method: String): byte[]
Client
client
input: Scanner
main(args: String[]): void
validateUser(id: String, user: User): String
EventType
functions
CONFERENCES: EventType
SEMINARS: EventType
TRADESHOWS: EventType
isValidEventType(eventype: String): boolean
Managers
client
LOGGER: Logger
user: User
input: Scanner
stub: EventSystemInterface
Managers(user: User)
run(): void
Options(): void
displayMenu(): int
setupLogging(): void
Customers
client
LOGGER: Logger
user: User
input: Scanner
stub: EventSystemInterface
Customers(user: User)
run(): void
Options(): void
displayMenu(): int
setupLogging(): void
User
client
city: City
role: Role
id: int
User()
User(city: City, role: Role, id: int)
getcity(): City
getRole(): Role
getId(): int
setcity(city: City): void
setRole(role: Role): void
setId(id: int): void
toString(): String
Role
functions
Customer: Role
Manager: Role
value: String
Role(value: String)
toString(): String
fromString(text: String): Role
City
functions
TOR: City
MTL: City
OTW: City
udpPort: int
City(udpPort: int)
getUdpPort(): int
cityExist(city: String): boolean
MTL_Server
server
LOGGER: Logger
main(args: String[]): void
setupLogging(): void
OTW_Server
server
LOGGER: Logger
main(args: String[]): void
setupLogging(): void
TOR_Server
server
LOGGER: Logger
main(args: String[]): void
setupLogging(): void
FuntionMembers
functions
validateUser(id: String, userRole: Role, city: City): SimpleEntry<Boolean,String>
validateEvent(evenId: String): SimpleEntry<Boolean,String>
validateEvent(evenId: String, city: City, time: String): SimpleEntry<Boolean,String>
validateType(eventType: String): SimpleEntry<Boolean,String>
objectToByteArray(obj: Object): byte[]
byteArrayToObject(data: byte[]): Object
cityMatch(city: String): boolean
roleMatch(role: String): boolean

# 4 Key Features

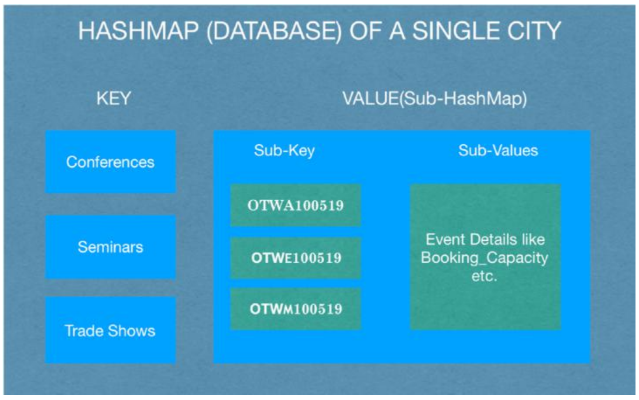
A. Data type for parameters

All the parameters passed are of type "String" and thus comply with the interface contract.

B. In-Memory database

Each server maintains its in-memory database. This database is implemented as:

$$HashMap < String, HashMap < String, HashMap < String, Object >> > cityDatabase$$



C. Case insensitive

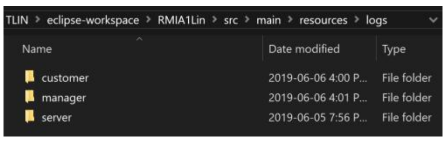
The user input is case insensitive， all the letters entered are case-insensitive. For users, this setting will make some operations easier.

D. Tread safe

Since each user runs on a thread. So, the system needs to be able to handle concurrent requests and provide thread safety to its data. This is achieved by using "***synchronized***" blocks whenever there is a need to update the in-memory database.

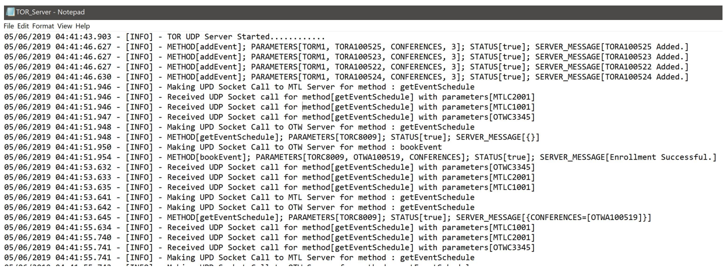
E. Logging

Custom logger is used to log all the messages. Each user operation is logged into the user specific log file. The same is true for departmental servers too.



TLIN › eclipse-workspace › RMIA1Lin › src › main › resources › logs

| Name | Date modified | Type |
|---|---|---|
| customer | 2019-06-06 4:00 P... | File folder |
| manager | 2019-06-06 4:01 P... | File folder |
| server | 2019-06-05 7:56 P... | File folder |



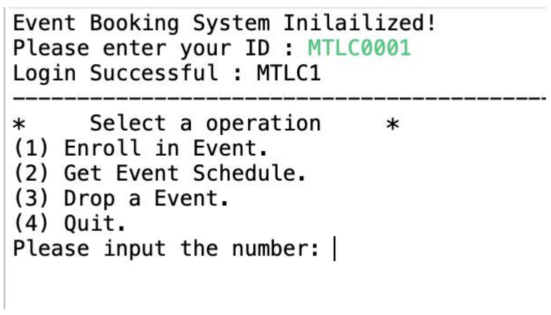
TOR_Server - Notepad

```
05/06/2019 04:41:43.903 - [INFO] - TOR UDP Server Started............
05/06/2019 04:41:46.627 - [INFO] - METHOD[addEvent]; PARAMETERS[TORM1, TORA100525, CONFERENCES, 3]; STATUS[true]; SERVER_MESSAGE[TORA100525 Added.]
05/06/2019 04:41:46.627 - [INFO] - METHOD[addEvent]; PARAMETERS[TORM1, TORA100523, CONFERENCES, 3]; STATUS[true]; SERVER_MESSAGE[TORA100523 Added.]
05/06/2019 04:41:46.627 - [INFO] - METHOD[addEvent]; PARAMETERS[TORM1, TORA100522, CONFERENCES, 3]; STATUS[true]; SERVER_MESSAGE[TORA100522 Added.]
05/06/2019 04:41:46.630 - [INFO] - METHOD[addEvent]; PARAMETERS[TORM1, TORA100524, CONFERENCES, 3]; STATUS[true]; SERVER_MESSAGE[TORA100524 Added.]
05/06/2019 04:41:51.946 - [INFO] - Making UPD Socket Call to MTL Server for method : getEventSchedule
05/06/2019 04:41:51.946 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[MTLC2001]
05/06/2019 04:41:51.946 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[MTLC1001]
05/06/2019 04:41:51.947 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[OTWC3345]
05/06/2019 04:41:51.948 - [INFO] - Making UPD Socket Call to OTW Server for method : getEventSchedule
05/06/2019 04:41:51.948 - [INFO] - METHOD[getEventSchedule]; PARAMETERS[TORC8009]; STATUS[true]; SERVER_MESSAGE[{}]
05/06/2019 04:41:51.950 - [INFO] - Making UPD Socket Call to OTW Server for method : bookEvent
05/06/2019 04:41:51.954 - [INFO] - METHOD[bookEvent]; PARAMETERS[TORC8009, OTWA100519, CONFERENCES]; STATUS[true]; SERVER_MESSAGE[Enrollment Successful.]
05/06/2019 04:41:53.632 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[OTWC3345]
05/06/2019 04:41:53.633 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[MTLC2001]
05/06/2019 04:41:53.635 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[MTLC1001]
05/06/2019 04:41:53.641 - [INFO] - Making UPD Socket Call to MTL Server for method : getEventSchedule
05/06/2019 04:41:53.642 - [INFO] - Making UPD Socket Call to OTW Server for method : getEventSchedule
05/06/2019 04:41:53.645 - [INFO] - METHOD[getEventSchedule]; PARAMETERS[TORC8009]; STATUS[true]; SERVER_MESSAGE[{CONFERENCES=[OTWA100519]}]
05/06/2019 04:41:55.634 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[MTLC1001]
05/06/2019 04:41:55.740 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[MTLC2001]
05/06/2019 04:41:55.741 - [INFO] - Received UDP Socket call for method[getEventSchedule] with parameters[OTWC3345]
05/06/2019 04:41:55.741 - [INFO] - Making UPD Socket Call to MTL Server for method : getEventSchedule
```

# 5 Test Cases

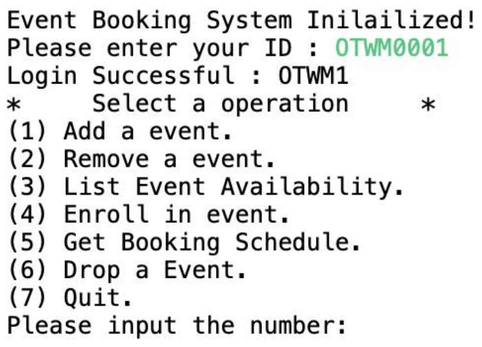
All cases has been tested to get correct result.

Customer options



```
Event Booking System Inilailized!
Please enter your ID : MTLC0001
Login Successful : MTLC1
-------------------------------------------
*      Select a operation      *
(1) Enroll in Event.
(2) Get Event Schedule.
(3) Drop a Event.
(4) Quit.
Please input the number: |
```

Manager options



```
Event Booking System Inilailized!
Please enter your ID : OTWM0001
Login Successful : OTWM1
*      Select a operation      *
(1) Add a event.
(2) Remove a event.
(3) List Event Availability.
(4) Enroll in event.
(5) Get Booking Schedule.
(6) Drop a Event.
(7) Quit.
Please input the number:
```

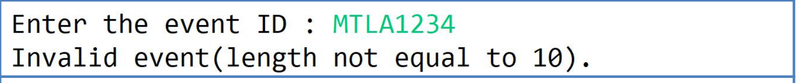
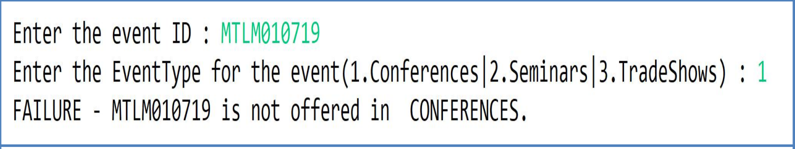
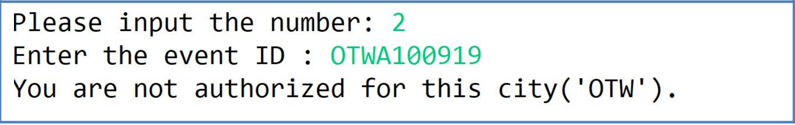
## 5.1 Login Check

| No. | Function | Screenshot |
|---|---|---|
| 1 | Check city | `Please enter your ID : SHEC0001`<br>`Your city('SHE') isn't recognized.` |
| 2 | Check User Type | `Please enter your ID : MTLH1111`<br>`Your role('H') isn't recognized.` |
| 3 | Check ID number | `Please enter your ID : OTWC1j2d`<br>`Your id('1j2d') isn't recognized.` |

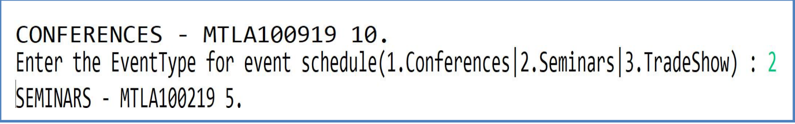
## 5.2 Manager Add Event

| No. | Function | Screenshot |
|---|---|---|
| 1 | Check event ID | `Enter the event ID : MTLA10019`<br>`Invalid event(length not equal to 10).` |
| 2 | Add event to another city | `Enter the event ID : OTWA081119`<br>`You are not authorized for this city('OTW').` |
| 3 | Check event's time format | `Enter the event ID : TORA091819`<br>`Invalid month (Should be 1-12)` |
| 4 | Check event's time char | `Enter the event ID : MTLW111119`<br>`Invalid time slots [4th char not equal to Morning(M), Afternoon(A) and Evening(E)].` |
| 5 | Adding duplicate event | `Enter the event ID : MTLA100919`<br>`Event Capacity : 10`<br>`Enter the EventType for the event(1.Conferences\|2.Seminars\|3.TradeShows) : 1`<br>`FAILURE = MTLA100919 is already offered in CONFERENCES, only capacity 10 would be updated.` |

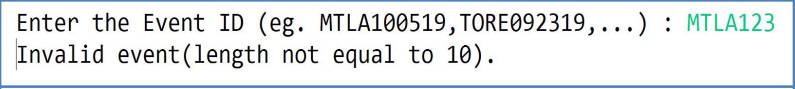
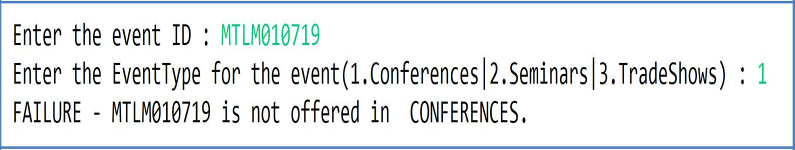
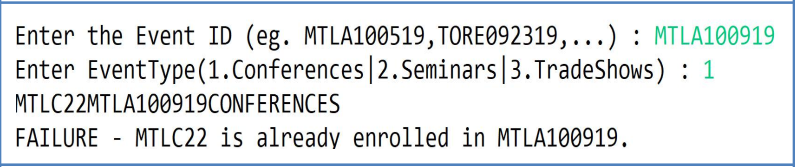
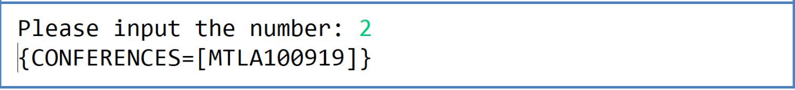
## 5.3 Manager Remove Event

| No. | Function | Screenshot |
|---|---|---|
| 1 | Check event ID |  Enter the event ID : MTLA1234<br>Invalid event(length not equal to 10). |
| 2 | Event doesn’t exist |  Enter the event ID : MTLM010719<br>Enter the EventType for the event(1.Conferences\|2.Seminars\|3.TradeShows) : 1<br>FAILURE - MTLM010719 is not offered in  CONFERENCES. |
| 3 | Remove another city’s event |  Please input the number: 2<br>Enter the event ID : OTWA100919<br>You are not authorized for this city('OTW'). |

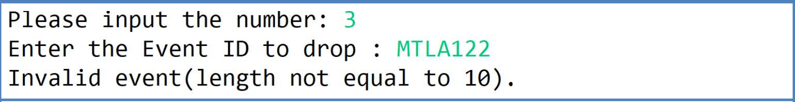
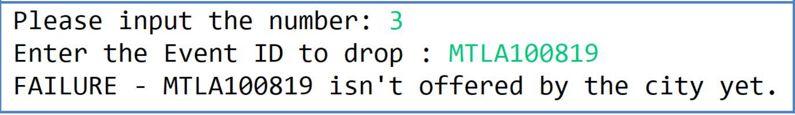
## 5.4 Manager List Event Availability

| No. | Function | Screenshot |
|---|---|---|
| 1 | List Availability |  CONFERENCES - MTLA100919 10.<br>Enter the EventType for event schedule(1.Conferences\|2.Seminars\|3.TradeShow) : 2<br>SEMINARS - MTLA100219 5. |

## 5.5 Customer Booking Event

| No. | Function | Screenshot |
|---|---|---|
| 1 | Check event ID |  Enter the Event ID (eg. MTLA100519,TORE092319,...) : MTLA123<br>Invalid event(length not equal to 10). |
| 2 | Event doesn’t exist |  Enter the event ID : MTLM010719<br>Enter the EventType for the event(1.Conferences\|2.Seminars\|3.TradeShows) : 1<br>FAILURE - MTLM010719 is not offered in  CONFERENCES. |
| 3 | Already enrolled |  Enter the Event ID (eg. MTLA100519,TORE092319,...) : MTLA100919<br>Enter EventType(1.Conferences\|2.Seminars\|3.TradeShows) : 1<br>MTLC22MTLA100919CONFERENCES<br>FAILURE - MTLC22 is already enrolled in MTLA100919. |
| 4 | List booked |  Please input the number: 2<br>{CONFERENCES=[MTLA100919]} |

## 5.6 Customer Drop Event

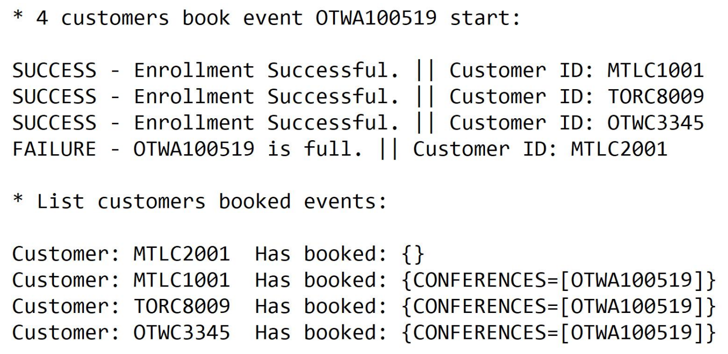
| No. | Function | Screenshot |
|---|---|---|
| 1 | Check event ID |  Please input the number: 3<br>Enter the Event ID to drop : MTLA122<br>Invalid event(length not equal to 10). |
| 2 | Event doesn’t exist |  Please input the number: 3<br>Enter the Event ID to drop : MTLA100819<br>FAILURE - MTLA100819 isn't offered by the city yet. |

## 5.7 Multi-Threads Test

In order to test the robustness of the system more conveniently, we write multi-threaded test code, which creates 12 events, four customers and two managers for three servers.

We also designed four test cases to simulate the results.

**Case1: 4 customers(from different server) book same event(3 slots) at same time.**



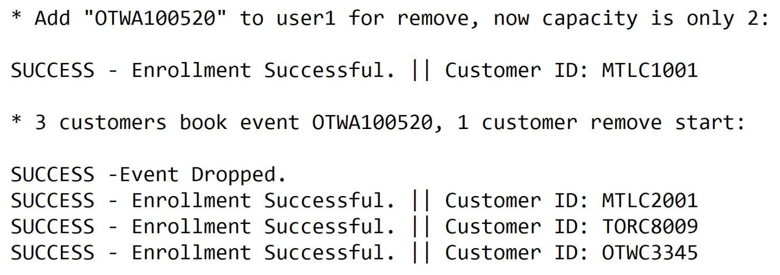
```
* 4 customers book event OTWA100519 start:

SUCCESS - Enrollment Successful. || Customer ID: MTLC1001
SUCCESS - Enrollment Successful. || Customer ID: TORC8009
SUCCESS - Enrollment Successful. || Customer ID: OTWC3345
FAILURE - OTWA100519 is full. || Customer ID: MTLC2001

* List customers booked events:

Customer: MTLC2001  Has booked: {}
Customer: MTLC1001  Has booked: {CONFERENCES=[OTWA100519]}
Customer: TORC8009  Has booked: {CONFERENCES=[OTWA100519]}
Customer: OTWC3345  Has booked: {CONFERENCES=[OTWA100519]}
```

**Case2: 1 user remove, other 3 try to book to event (slots only 3).**



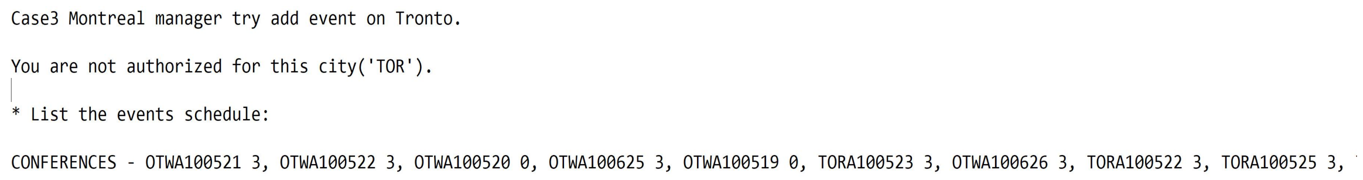
```
* Add "OTWA100520" to user1 for remove, now capacity is only 2:

SUCCESS - Enrollment Successful. || Customer ID: MTLC1001

* 3 customers book event OTWA100520, 1 customer remove start:

SUCCESS -Event Dropped.
SUCCESS - Enrollment Successful. || Customer ID: MTLC2001
SUCCESS - Enrollment Successful. || Customer ID: TORC8009
SUCCESS - Enrollment Successful. || Customer ID: OTWC3345
```

**Case3: Montreal manager try adding event on Toronto.**



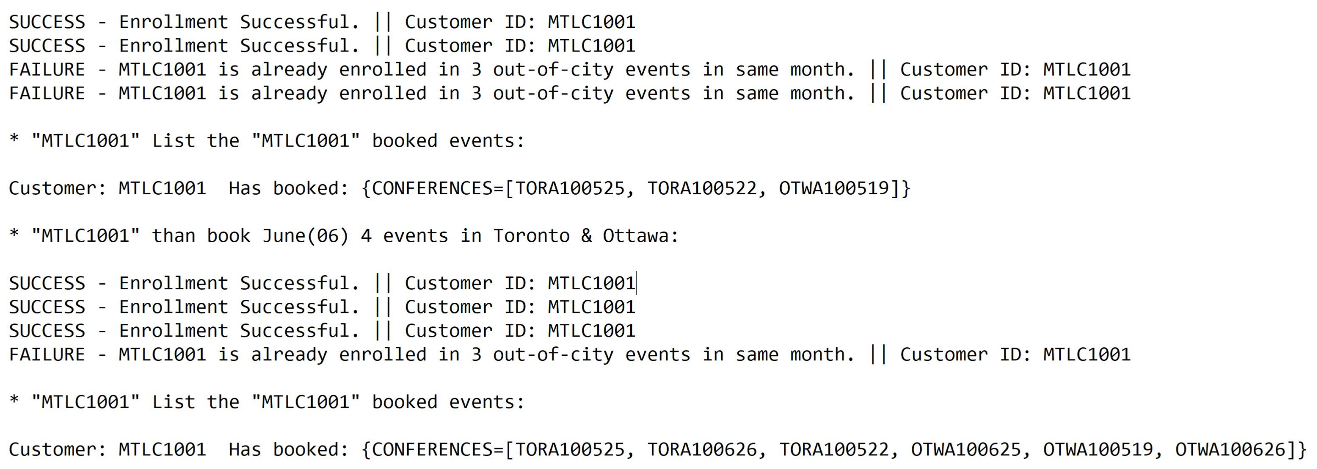
```
Case3 Montreal manager try add event on Tronto.

You are not authorized for this city('TOR').

* List the events schedule:

CONFERENCES - OTWA100521 3, OTWA100522 3, OTWA100520 0, OTWA100625 3, OTWA100519 0, TORA100523 3, OTWA100626 3, TORA100522 3, TORA100525 3,
```

**Case4: Montreal customer try book 4 events on Toronto in same month.**



```
SUCCESS - Enrollment Successful. || Customer ID: MTLC1001
SUCCESS - Enrollment Successful. || Customer ID: MTLC1001
FAILURE - MTLC1001 is already enrolled in 3 out-of-city events in same month. || Customer ID: MTLC1001
FAILURE - MTLC1001 is already enrolled in 3 out-of-city events in same month. || Customer ID: MTLC1001

* "MTLC1001" List the "MTLC1001" booked events:

Customer: MTLC1001  Has booked: {CONFERENCES=[TORA100525, TORA100522, OTWA100519]}

* "MTLC1001" than book June(06) 4 events in Toronto & Ottawa:

SUCCESS - Enrollment Successful. || Customer ID: MTLC1001
SUCCESS - Enrollment Successful. || Customer ID: MTLC1001
SUCCESS - Enrollment Successful. || Customer ID: MTLC1001
FAILURE - MTLC1001 is already enrolled in 3 out-of-city events in same month. || Customer ID: MTLC1001

* "MTLC1001" List the "MTLC1001" booked events:

Customer: MTLC1001  Has booked: {CONFERENCES=[TORA100525, TORA100626, TORA100522, OTWA100625, OTWA100519, OTWA100626]}
```

# REFERENCES:

1. https://www.tutorialspoint.com/java_rmi/java_rmi_application.htm
2. https://www.javatpoint.com/RMI
3. https://docs.oracle.com/javase/tutorial/rmi/index.html
4. https://www.mkyong.com/java/java-rmi-distributed-objects-example/
5. https://www.oreilly.com/library/view/learning-java-4th/9781449372477/ch13s04.html
6. https://stackoverflow.com/questions/36739915/sending-rmi-stub-over-udp
7. https://cs.nyu.edu/courses/fall02/G22.3033-009/Lectures/lecture%20006.pdf