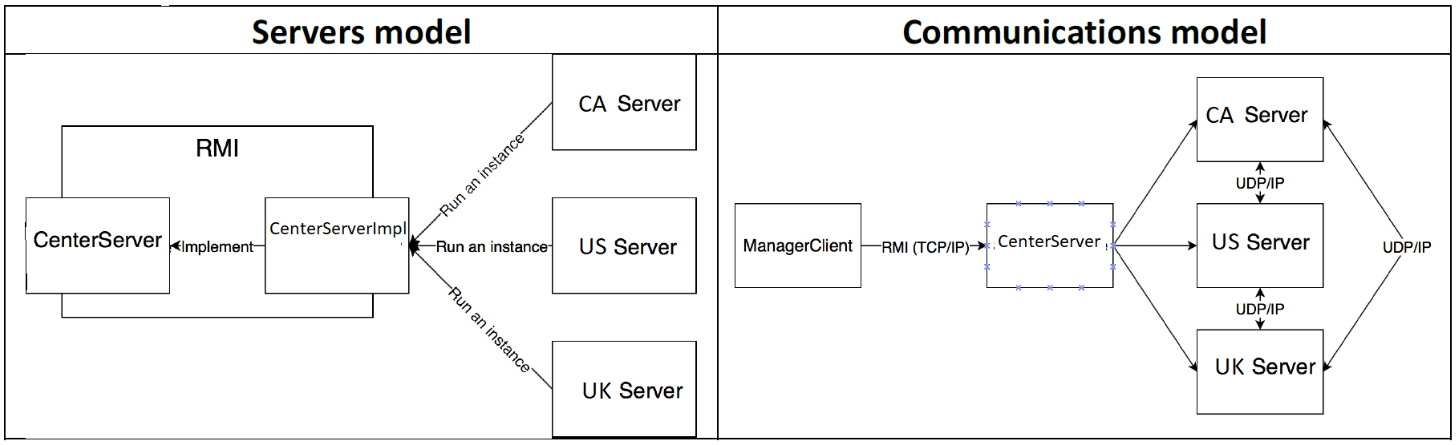


CenterServer.java

public interface CenterServer

- The RMI interface

```
String createMRecord(String firstname, String lastname,String employeeID ,String emailID ,ArrayList<String> projectInfo ,String location) throws RemoteException, ServerNotActiveException;
```

- Method to create manager record.

```
String createERecord(String firstname, String lastname,String employeeID ,String emailID, String projectID) throws RemoteException, ServerNotActiveException;
```

-Method to create Employee Record

```
String getRecordCounts() throws RemoteException;
```

- Method to get records of each server and returns to user.

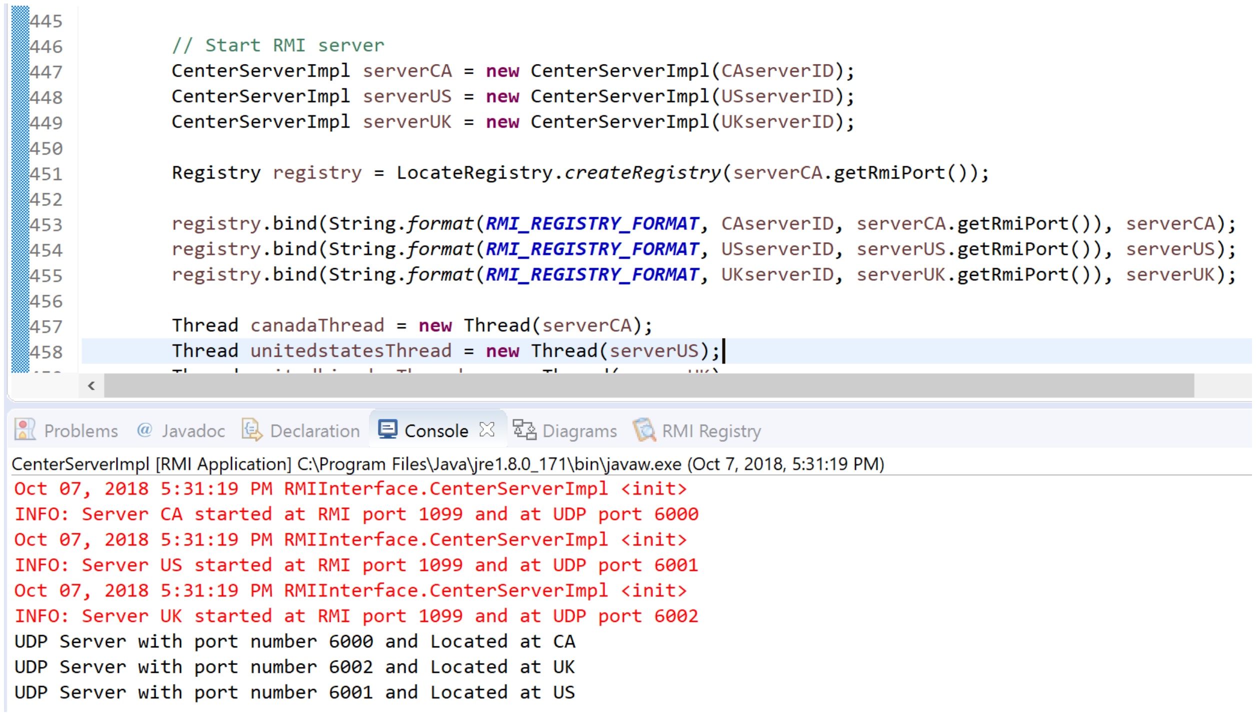
```
boolean editRecord(String recordID,String fieldName,String newValue) throws RemoteException;
```

-Method to edit records of employee or manager.

```
void printData() throws RemoteException;
```

-Prints the information stored when employee or manager created.

Starting the server:



```
        // Start RMI server
        CenterServerImpl serverCA = new CenterServerImpl(CAserverID);
        CenterServerImpl serverUS = new CenterServerImpl(USserverID);
        CenterServerImpl serverUK = new CenterServerImpl(UKserverID);

        Registry registry = LocateRegistry.createRegistry(serverCA.getRmiPort());

        registry.bind(String.format(RMI_REGISTRY_FORMAT, CAserverID, serverCA.getRmiPort()), serverCA);
        registry.bind(String.format(RMI_REGISTRY_FORMAT, USserverID, serverUS.getRmiPort()), serverUS);
        registry.bind(String.format(RMI_REGISTRY_FORMAT, UKserverID, serverUK.getRmiPort()), serverUK);

        Thread canadaThread = new Thread(serverCA);
        Thread unitedstatesThread = new Thread(serverUS);
```

```
CenterServerImpl [RMI Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (Oct 7, 2018, 5:31:19 PM)
Oct 07, 2018 5:31:19 PM RMIInterface.CenterServerImpl <init>
INFO: Server CA started at RMI port 1099 and at UDP port 6000
Oct 07, 2018 5:31:19 PM RMIInterface.CenterServerImpl <init>
INFO: Server US started at RMI port 1099 and at UDP port 6001
Oct 07, 2018 5:31:19 PM RMIInterface.CenterServerImpl <init>
INFO: Server UK started at RMI port 1099 and at UDP port 6002
UDP Server with port number 6000 and Located at CA
UDP Server with port number 6002 and Located at UK
UDP Server with port number 6001 and Located at US
```

The Server

The server starts by creating 3 separate objects for the 3 different servers, each will bind with the registry that is connected to RMI port 1099. This will make all three server start at the same time.

Next I created threads so I can use semaphores and lock the id's making sure no duplicate id's exists.

The connected I used is a UDP connection, that connects to three different ports, 6000, 6001, and 6002. Each server is connected to its own server.

The Client

A

```
Enter your ManagerID
DA
Invalid Manager ID!! Manager Id should start with CA or US or UK
Enter your ManagerID
CA123
Choose from the  following options
0 Print the options again
1 Create Employee Records
2 Create Manager Records
3 Edit the Records
4 Get the Record Count
5 Exit
Enter Your Choice
```

The application startup begins with a manager login, the application will warn the user if the ID doesn't start with CA, US or UK. To login, you can use any number after the Server ID. At the moment I haven't specified the Manager ID combo so as long as CA, US or UK is in the starting Manager ID, login will be successful.

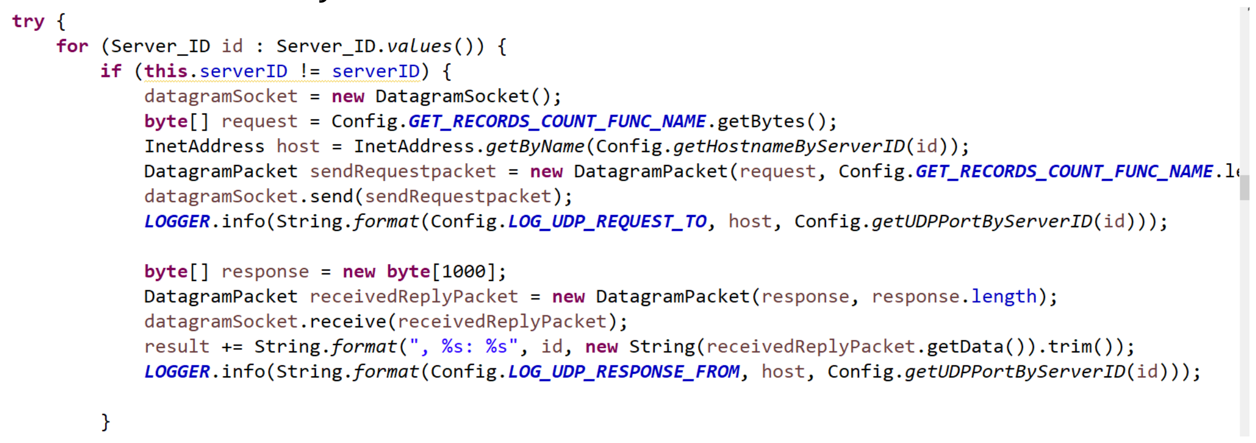
The choices:

1) View the options again
   This options will reprint the options that manager can do.
2) Create an Employee Record
   This will invoke the create Employee record function from the Interface. It will create an employee, with a unique employee ID starting with ER and store it in the file of the manager who created it.

3) Create a Manager Record
   This will invoke the create Manager record function from the Interface. It will create a manager, with a unique manager ID starting with MR and store it in the file of the manager who created it.

4) Edit a Record
   This will allow the manager to edit the mutable fieldnames available for either an employee or a manager.

5) View the Record Counts
This will show the number of records of each server. It will check the records in the current UDP port example:
If we are a Canadian manager, our UDP port is 6000, to get the record of the other servers (6001, and 6002) we must connect the those UDP's as well. Which means that after we got our record count, we must connect to the UDP ports that are not ours so that we can see how many records exists.



```
try {
    for (Server_ID id : Server_ID.values()) {
        if (this.serverID != serverID) {
            datagramSocket = new DatagramSocket();
            byte[] request = Config.GET_RECORDS_COUNT_FUNC_NAME.getBytes();
            InetAddress host = InetAddress.getByName(Config.getHostnameByServerID(id));
            DatagramPacket sendRequestpacket = new DatagramPacket(request, Config.GET_RECORDS_COUNT_FUNC_NAME.l
            datagramSocket.send(sendRequestpacket);
            LOGGER.info(String.format(Config.LOG_UDP_REQUEST_TO, host, Config.getUDPPortByServerID(id)));

            byte[] response = new byte[1000];
            DatagramPacket receivedReplyPacket = new DatagramPacket(response, response.length);
            datagramSocket.receive(receivedReplyPacket);
            result += String.format(", %s: %s", id, new String(receivedReplyPacket.getData()).trim());
            LOGGER.info(String.format(Config.LOG_UDP_RESPONSE_FROM, host, Config.getUDPPortByServerID(id)));

        }
```

A User Datagram Protocol, has a request and a response the request, will request the record count from another Server (different UDP address) The active server will start a new thread to handle the request. This implementation lets the server being able to handle multiple request concurrently, quickly come back to listening to avoid missing any requests. This will get as a response the number of records of the ports that is not its own.

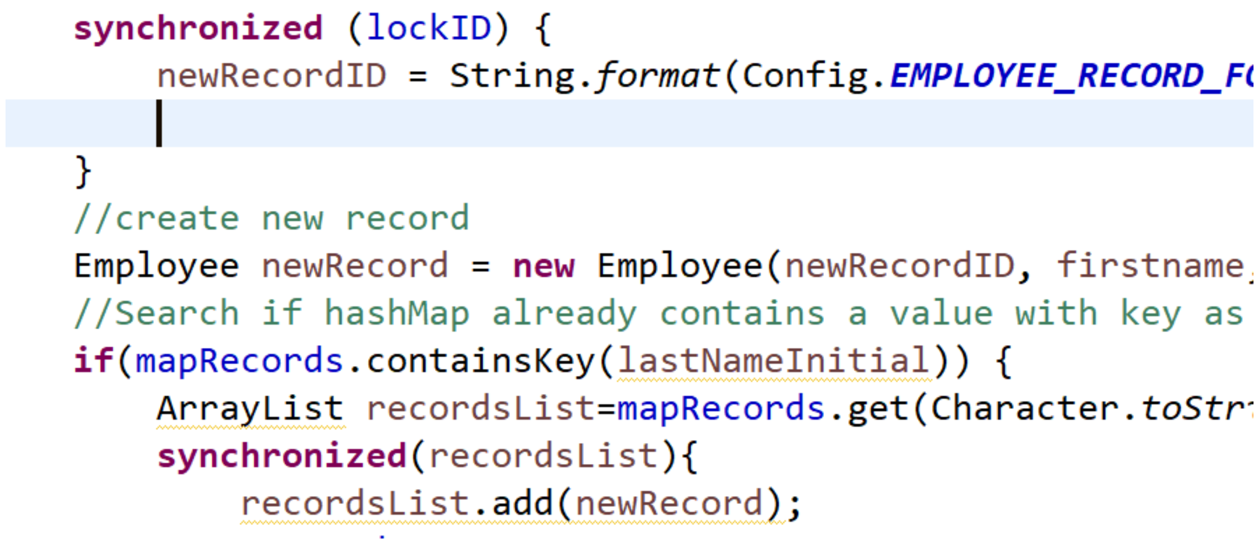
6) Quit (logout)
This will log the manager out and will allow another manager to login in to complete its task. I did it this way so that it is easier to test without opening the application in another console.

<u>Problems / Issues</u>

The problems I encountered while doing this assignment was that I had forgot to switch servers when I chose to logout. Or logged in with a new manager. This caused the records count to always go into the same folder and the manager logs were getting overwritten every time. I fixed this issue by closing the file after the manager logs out. By closing the file, you can no longer write to that file. Thus the issue of registering employee or manager into the wrong server was fixed.

Threading and Locks

The important part of this assignment was to lock the record Id of newly created employee or manager. I was able to do this by use Locks from multithreading.



```
synchronized (lockID) {
    newRecordID = String.format(Config.EMPLOYEE_RECORD_F
    
}
//create new record
Employee newRecord = new Employee(newRecordID, firstname
//Search if hashMap already contains a value with key as
if(mapRecords.containsKey(lastNameInitial)) {
    ArrayList recordsList=mapRecords.get(Character.toStr
    synchronized(recordsList){
        recordsList.add(newRecord);
```

Synchronized lockID, locks the record ID once it has been initialized. Therefore, for Example: if ER00000 has been created, we will lock the record id, so that no other records will have the same id. After which each record id will increment by 1, after the initial record has been created.

Another lock I used was a lock object called lockCount, this will lock the records count once it has been requested by the UDP port for the get records count part. Since we access different UDP ports we need to lock it to make sure that it wont change as we fetch the count.