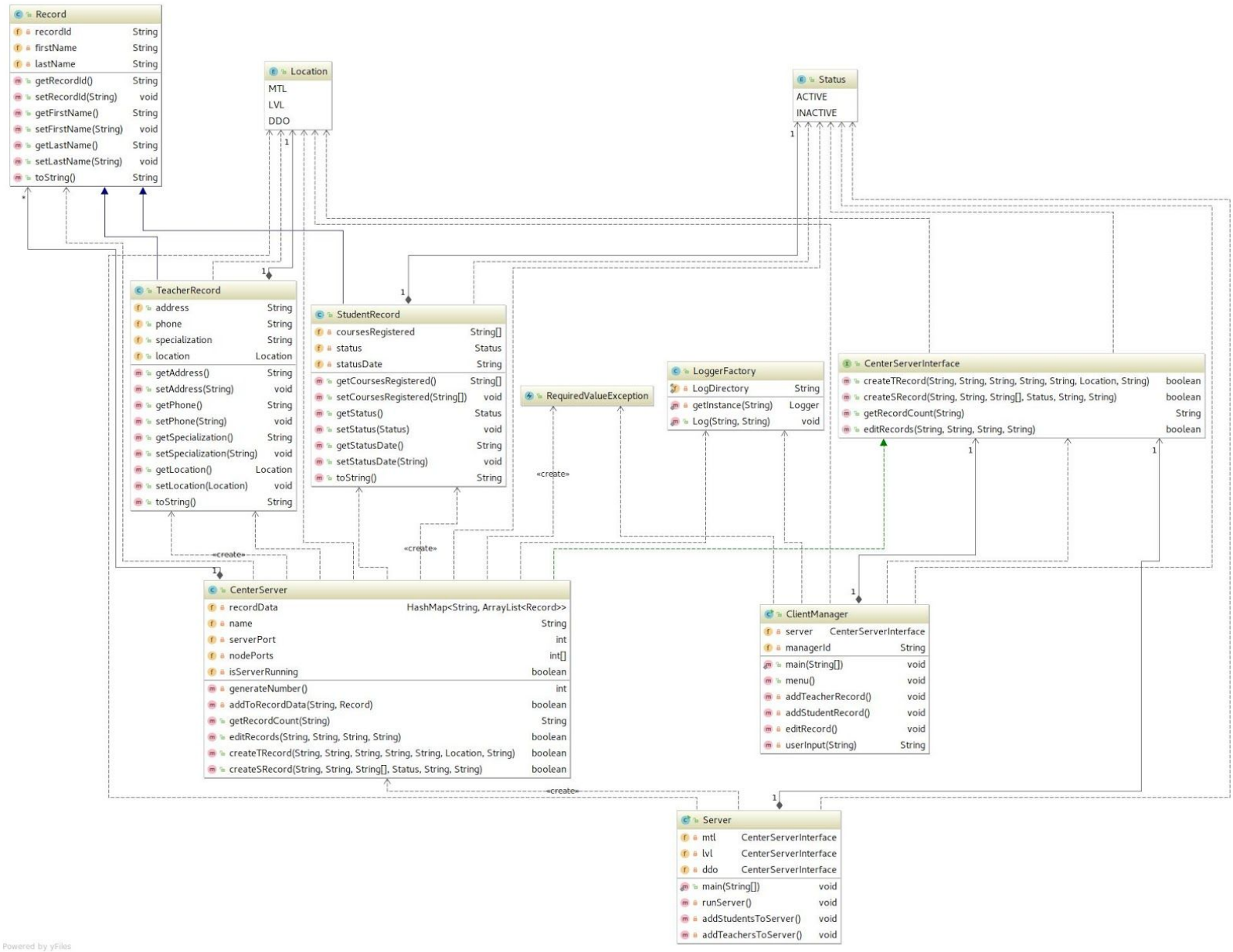
Assignment 1

Architecture:

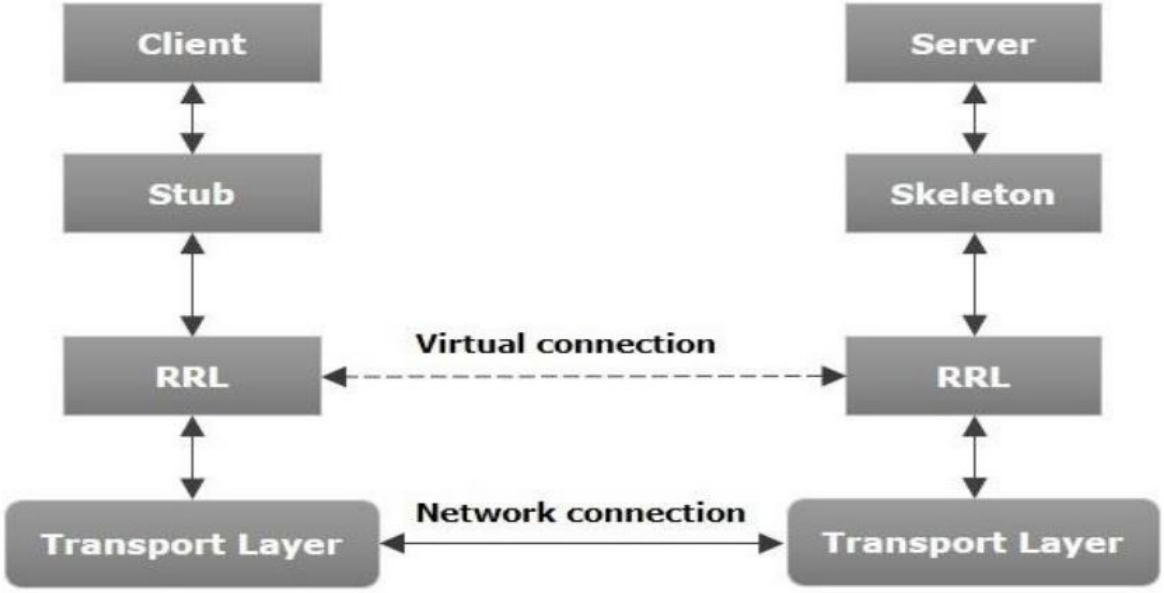
- **Interface(CenterServerInterface.java) :** Java RMI Interface containing methods like createSRecord, createTRecord, getRecordCounts, editRecord. This interface extends the Remote class. Each method throws two types of exception, one is RemoteException and another one is our customized exception which is RequiredValueException which is thrown when any of the input validations fails.
- **CenterServer :** This class implements the interface mentioned above and extends UniCastRemoteObject class. There are 3 center servers for Montreal, Laval and DDO. Each has their own hashmap to store the student and teacher record. Initially there are few records for student and teacher. Hashmap maintains key and value pairs.
    - Here we have used Hashmap<String, ArrayList<Record>> where String as a key will contain alphabet between A to Z. Arraylist stores the list of students and teachers records according the first character of the last name. e.g. all the records whose last name starts with “A” will be stored in arraylist “a” and then put into hashmap with key “A”.
- **Server :** This is the main class for the server invocation. Server class extends CenterServer class. runServer() method will create 3 server instances which will run on 3 different ports and, will create the registry binding all the servers to it. Along with it, some default records are added into each server for testing purposes.
- **ClientManager :** It is a client which invokes the center’s server system to test the correct operation of the DCMS invoking CenterServer.
- **LoggerFactory :** It to generate the log of the activities performed during the execution of the program.

Following class diagram best describes our architecture that we have used :


Record
recordId String
firstName String
lastName String
getRecordId() String
setRecordId(String) void
getFirstName() String
setFirstName(String) void
getLastName() String
setLastName(String) void
toString() String
Location
MTL
LVL
DDO
Status
ACTIVE
INACTIVE
TeacherRecord
address String
phone String
specialization String
location Location
getAddress() String
setAddress(String) void
getPhone() String
setPhone(String) void
getSpecialization() String
setSpecialization(String) void
getLocation() Location
setLocation(Location) void
toString() String
StudentRecord
coursesRegistered String[]
status Status
statusDate String
getCoursesRegistered() String[]
setCoursesRegistered(String[]) void
getStatus() Status
setStatus(Status) void
getStatusDate() String
setStatusDate(String) void
toString() String
RequiredValueException
LoggerFactory
LogDirectory String
getInstance(String) Logger
Log(String, String) void
CenterServerInterface
createTRecord(String, String, String, String, String, Location, String) boolean
createSRecord(String, String, String[], Status, String, String) boolean
getRecordCount(String) String
editRecords(String, String, String, String) boolean
CenterServer
recordData HashMap<String, ArrayList<Record>>
name String
serverPort int
nodePorts int[]
isServerRunning boolean
generateNumber() int
addToRecordData(String, Record) boolean
getRecordCount(String) String
editRecords(String, String, String, String) boolean
createTRecord(String, String, String, String, String, Location, String) boolean
createSRecord(String, String, String[], Status, String, String) boolean
ClientManager
server CenterServerInterface
managerId String
main(String[]) void
menu() void
addTeacherRecord() void
addStudentRecord() void
editRecord() void
userInput(String) String
Server
mtl CenterServerInterface
lvl CenterServerInterface
ddo CenterServerInterface
main(String[]) void
runServer() void
addStudentsToServer() void
addTeachersToServer() void
«create»
«create»
«create»
«create»
1
*

Techniques Used:

- **Multithreading:** We are using Threads for parallel request to get the record counts from servers. One server will create appropriate threads and sends the request for getting record counts to remaining servers and will wait for their response.
- **Socket Programming:** Here we are using UDP programming. It uses datagram packets that provide functions to make a packet to transmit, which includes array of bytes containing message, Length of message, Internet address, port number. Datagram sockets provides support to send receive UDP datagram packets. Inetaddress is used for server. UDP programming is required because server has to communicate with other servers bi-directionally.
- **RMI:** Any time we have a function that requires some large centralized computing power or some expensive resource (a huge database for example), but our output needs to be many places where such a workload can't be deployed then we would look at Remote Method Invocation.



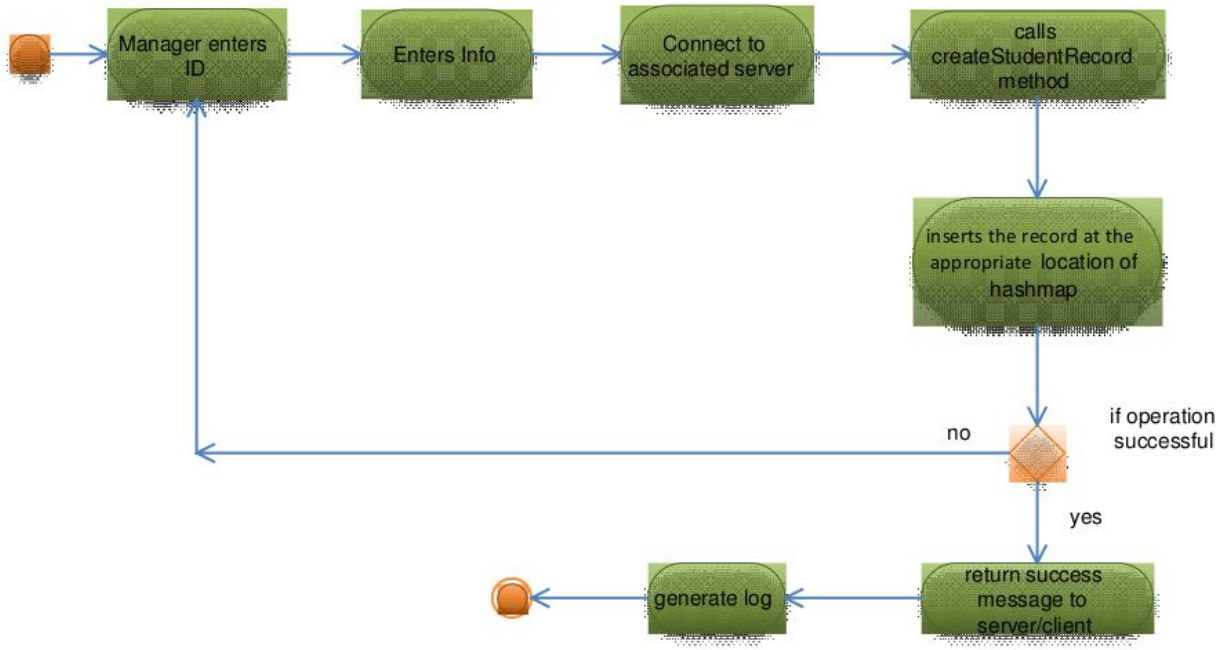
In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

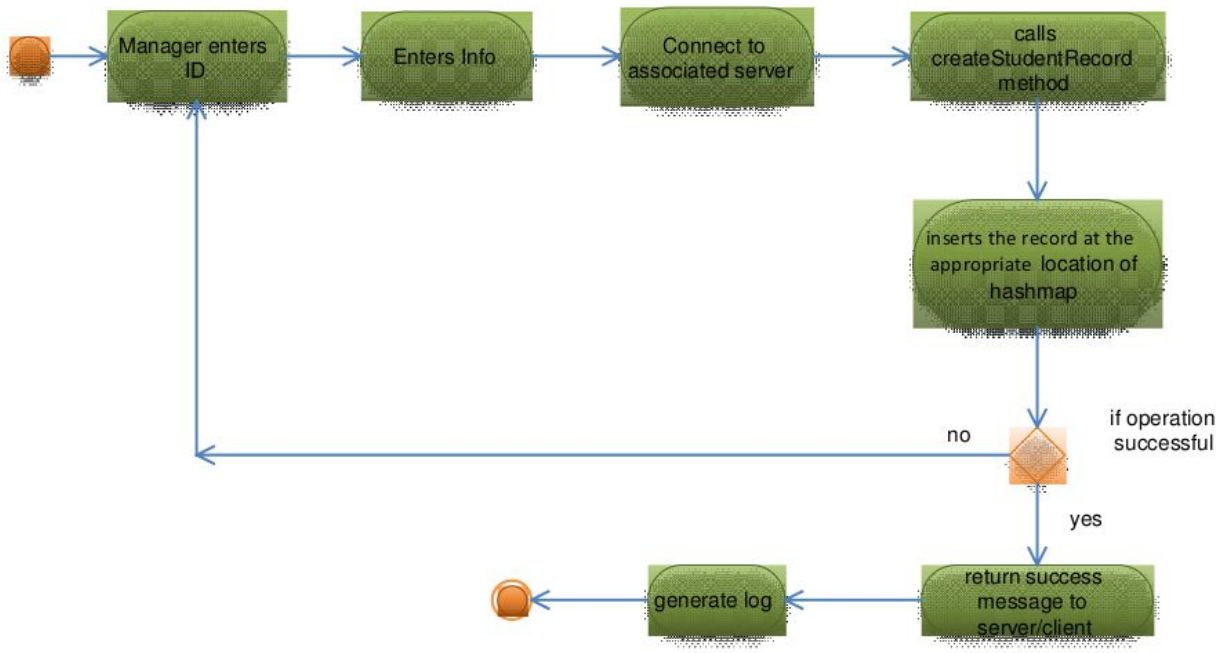
Test Scenarios:

- **Create Student Record**
  To test the method that is responsible to create the new student record. When a manager invokes this method from his/her center through a client program called ClientManager, the server associated with this manager (determined by the

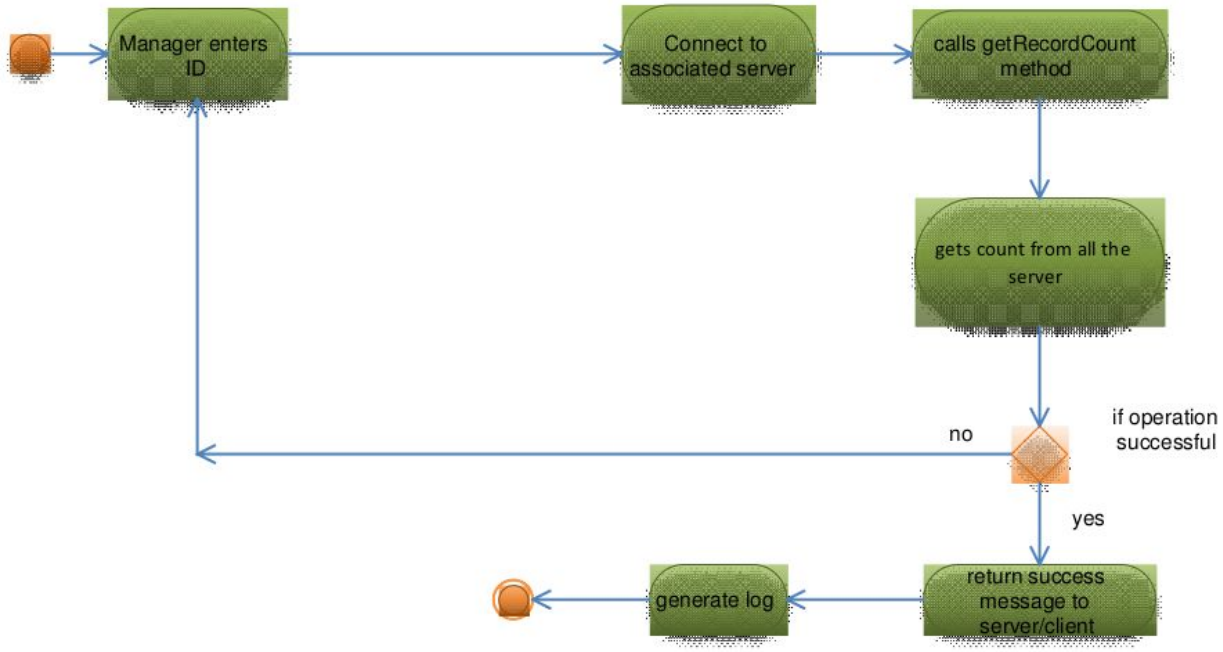
unique managerID prefix) attempts to create a TeacherRecord with the information passed, assigns a unique RecordID and inserts the Record at the appropriate location in the HashMap. The server returns information to the manager whether the operation was successful or not and both the server and the client store this information in their logs.



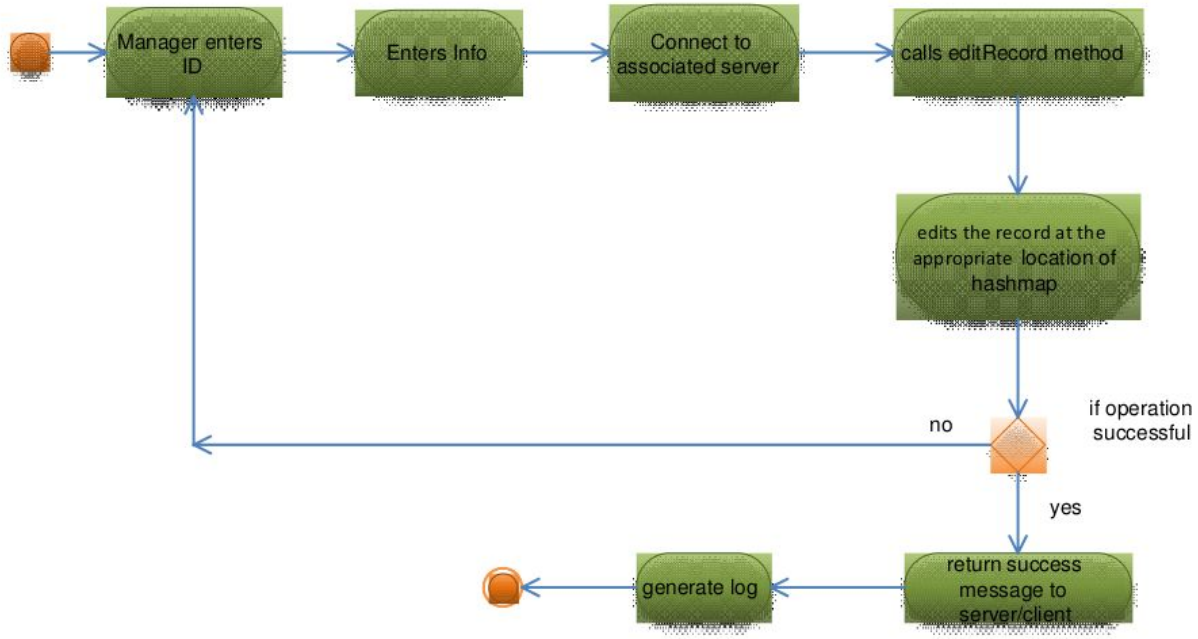
- **Create Teacher Record**
  To test the method that is responsible to create the new teacher record. When a manager invokes this method from a ClientManager, the server associated with this manager (determined by the unique managerID prefix) attempts to create a TeacherRecord with the information passed, assigns a unique RecordID and inserts the Record at the appropriate location in the hash map. The server returns information to the manager whether the operation was successful or not and both the server and the client store this information in their logs.



- **Get Record Counts**
  To test the method that is responsible to return the number of records stored on all servers. A manager invokes this method from his/her ClientManager and the server associated with that manager concurrently finds out the number of records (both TR and SR) in the other centers using UDP/IP sockets and returns the result to the manager. It only returns the record counts (a number) and not the
  records themselves. For example if MTL has 6 records, LVL has 7 and DDO had 8, it should return the following: MTL 6, LVL 7, DDO 8.



- **Edit Record**
  To test the method that is responsible to edit the record. When invoked by a manager, the server associated with this manager, (determined by the unique managerID) searches in the hash map to find the recordID and change the value of the field identified by “fieldname” to the newValue, if it is found. Upon success or failure it returns a message to the manager and the logs are updated with this information. If the new value of the fields such as location (Teacher), status (Student), does not match the type it is expecting, it is invalid. For example, if the found Record is a TeacherRecord and the field to change is location and newValue is other than mtl, lvl or ddo, the server shall return an error. The fields that should be allowed to change are address, phone and
  location (for TeacherRecord), and course registered, status and status date (for StudentRecord).



One the most important functionality that we have added to our project is the logger function. By developing such functionality we could able to keep track of any state changes in the project and it is helpful to debug the project to find errors.
Secord, the most difficult part for us to maximize the concurrency. We need to use some latch countdown and await mechanism to maximize the concurrency.
Another interesting section of the project is using the RMI in practice, we found RMI very powerful and easy to use, we have some thoughts about the performance issue of RMI in an enterprise environment. But there is no thought that we could develop the project fast and the configuration of RMI is easy. Now we have a clear perspective of RMI workflow by implementing simple commutations between server and clients.