Concordia University

COMP6231- Distributed System Design Assigment-1

Distributed Staff Management System (DSMS) using Java RMI

Date: 31st May, 2016.

Recipient: Prof. Sukhjinder Narula

Student Name: Yogesh Thakare(40013244)
Gaurav Amrutkar(40015917)

Table of Contents

1. OBJECTIVE

The main purpose of this assignment is to implement the Distributed Staff Management System (DSMS) using Java RMI. It is a distributed system used by clinic managers to manage information regarding the doctors and nurses across different clinics.

2. DATA STRUCTURES

2.1 UML DIAGRAMS

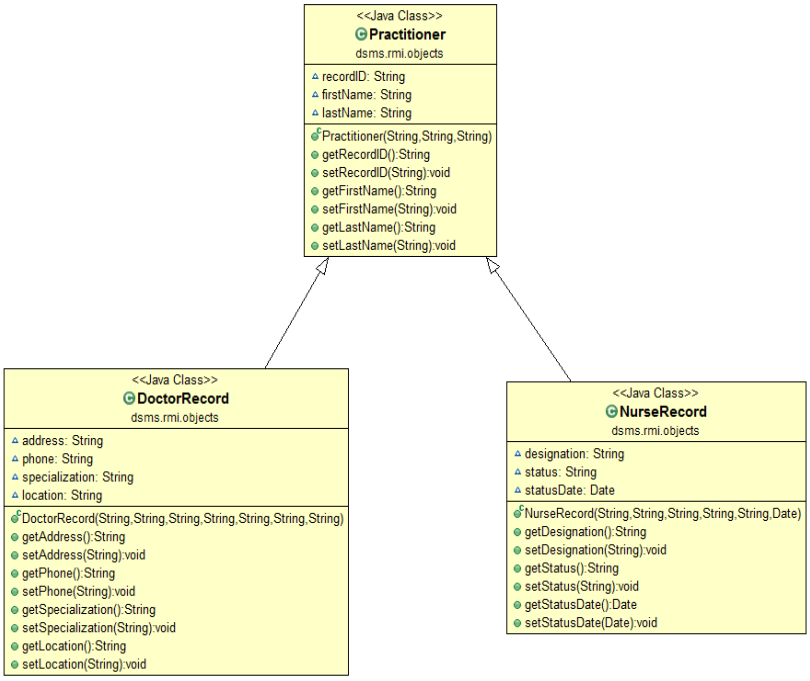


Figure1: Data Model

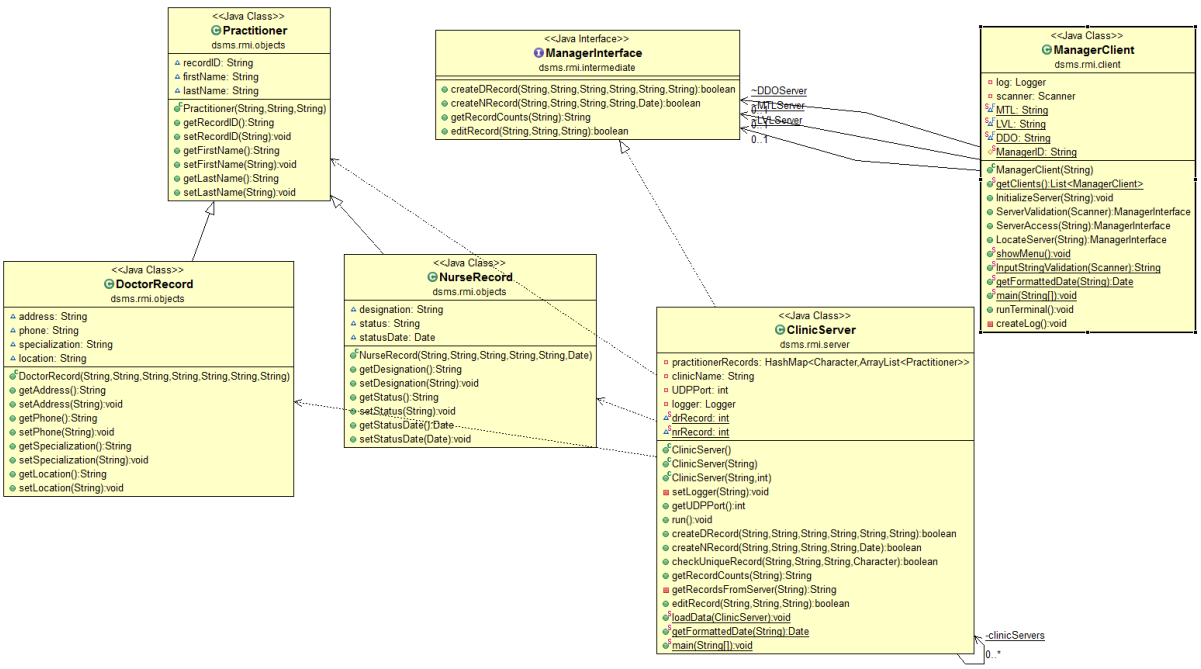


Figure 2: Class Diagrams

3. RMI OVERALL ARCHITECTURE

3.1 COMPONENTS

There are 3 basic components in RMI

1. Client:

It looks up for the remote object and once gets the access; it calls the methods on the obtained remote object.

2. Server:

Server registers itself in the RMI registry and accepts the method invocations from the client.

3. RMI Registry:

The registry is a remote object lookup service. The registry may run on the same host as the server or on a different host.

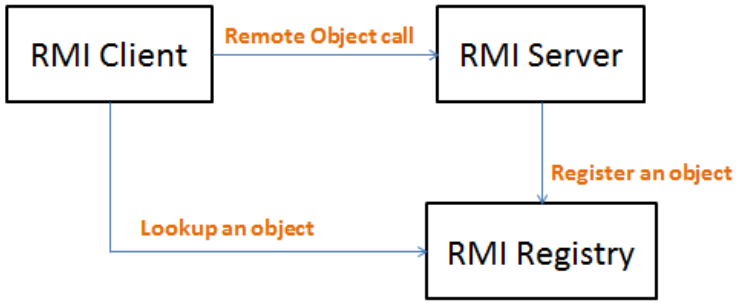
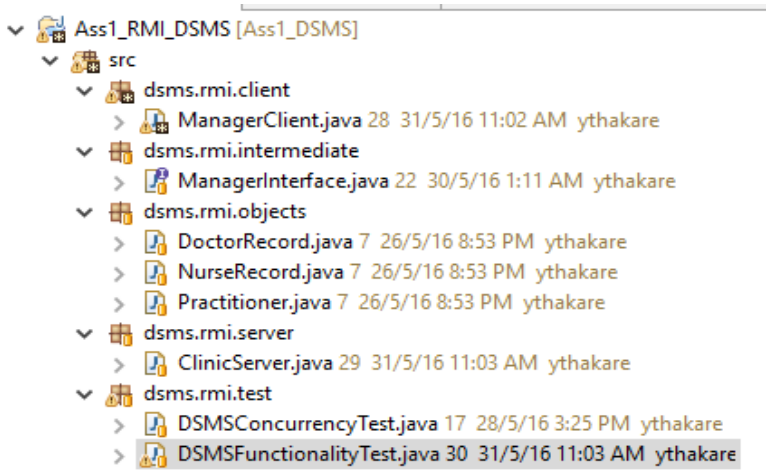


Figure: RMI Components

In Our Project:

Below is the structure of the project implemented in java using RMI architecture:



Here:

- RMI Client is – ManagerClient which invokes the ClinicServer class.
- ClientServer class implements ManagerInterface where all the below operations are defined:
  - ✓ createDRecord (firstName, lastName, address, phone, specialization, location)
  - ✓ createNRecord (firstName, lastName, designation, status, statusDate)
  - ✓ getRecordCounts (recordType)
  - ✓ editRecord (recordID, fieldName, newValue)
- Practitioner class is made as a parent class for DoctorRecord and NurseRecord so that Practitioner will able to accept the objects of DoctorRecord and NurseRecord both.

- In Hashmap, we are storing a list of Practitioner objects as a value against key as a character of English alphabets(A-Z).

3.2 DATA FLOW

Flow in RMI is as follow:

1) Server registers its server object in the RMI registry.
2) Client then look in the RMI registry
3) And gets the server object as server stub in client side.
4) Client invokes stub method
5) Stub talks to skeleton
6) Skeleton invokes the remote object method.

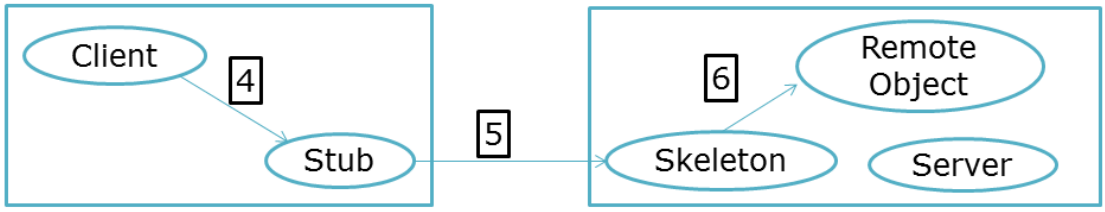


Figure: RMI Data Flow

4. MULTITHREADING AND CONCURRENCY

Concurrency is related closely with parallel computing i.e.“Multiple programs executing parallel at same time”

Concurrency means if a long time consuming task can be performed asynchronously or in parallel, which basically improves throughput of the program and its interactivity.

Process and Threads :

A process runs independently and isolated of other process. It cannot directly access the shared data from another process.

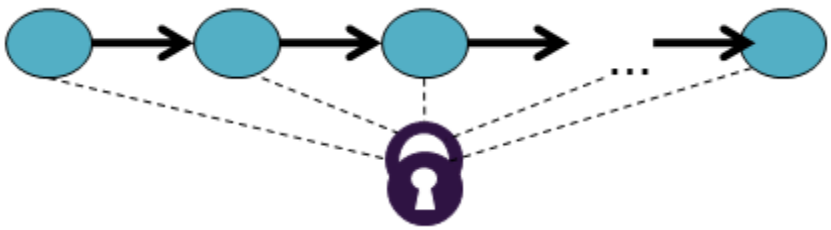
Thread is a lightweight process which has its own stack. Unlike process, a thread can acess the shared data of other threads in the same process. Thread stores the shared data in its own memory cache.

A typical java application runs as a single process.

4.1 FINE GRAIN LOCKING VS COARSE GRAINED LOCKING:

COARSE-GRAINED LOCKING:

- Lesser locks (One lock has more objects)



Example:

One lock for all the records in HashMap

Advantages:

- It is simple to implement.
- Faster/easier to implement operations that access multiple locations

FINE-GRAINED LOCKING:

- More locks (One lock has fewer objects)

Example:

One lock per record in HashMap in our practitioner record HashMap.

Advantages:

- More simultaneous access (performance improvement when coarse-grained would lead to unnecessary blocking).

Use in our project:

We have extended ClinicServer and ManagerClient by thread object(java.lang.Thread). So that, both of these becomes object of thread automatically.

We have created multiple instances of ClinicServer and ManagerClient in order to show multithreading demo.

Whenever we call start() method on these threads, the run() method get called automatically and the execution flow/time decided by JVM.

4.2 SYNCHRONIZATION:

Synchronization is a concept in java, which we use when two, or more threads try to access same resource simultaneously. It is a mechanism to coordinate the access to a common data and critical code among multiple threads. Java uses the locking mechanism to implement this coordination.

Synchronization concept can be explained with the help of below example as locking:

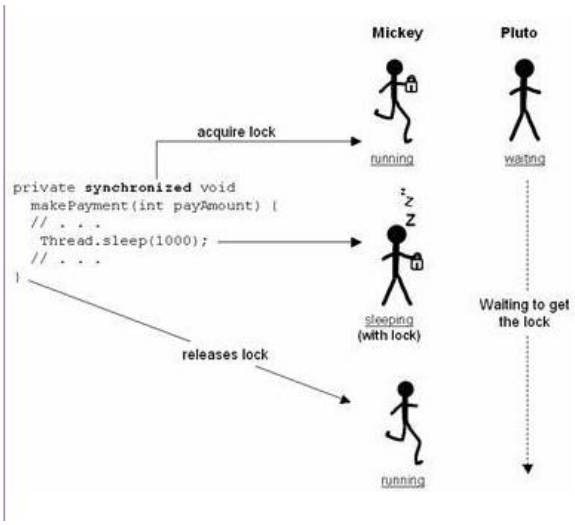


Figure: Synchronization

Synchronization can be implemented with the help of following:

- Synchronization methods
- Synchronization blocks **(used in our project)**

4.3 HASHMAP

HashMap in Java works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time provided we know the key. In hashing, hash functions are used to link key and key value in HashMap.

Objects are stored by calling put(key, value) method of HashMap and retrieved by calling get(key) method.

Use In Our project:

```
private HashMap<Character, ArrayList<Practitioner>> practitionerRecords =
                    new HashMap<Character, ArrayList<Practitioner>>();
```

We have used Hasmap to store the char and Arraylist of practitioner. We are storing a list of Practitioner objects as a value against key as a character of English alphabets.

We have used a concept of synchronized method while adding a new DR or NR record. We are storing this record first in practitioner list and then adding it to the practitionerRecords hashmap.:

```
synchronized(practitionerRecords)
{
ArrayList<Practitioner> practitionerList = practitionerRecords.get(lastName.charAt(0));
   if(practitionerList == null &&
checkUniqueRecord(Practitioner.getRecordID(),Practitioner.getFirstName(),Practit
ioner.getLastName(),lastName.charAt(0)))
   {
      practitionerList = new ArrayList<Practitioner>();
      practitionerList.add(Practitioner);
      practitionerRecords.put(lastName.charAt(0), practitionerList);
   }
   else
  if(checkUniqueRecord(Practitioner.getRecordID(),Practitioner.getFirstName(),P
  ractitioner.getLastName(),lastName.charAt(0)))
  {
      practitionerList.add(Practitioner);

  }
  else
```

```
 {
      logger.info("Failed to add Nurse Record with record ID :
"+Practitioner.getRecordID()+" duplicate record ID");
                        return false;
  }
```

5. UDP IMPLEMENTATION

- UDP : No connection between client and server.
- No handshaking.
- Sender explicitly attaches IP address of destination to each packet.
- Server must extract IP address, port of sender from receive packet.

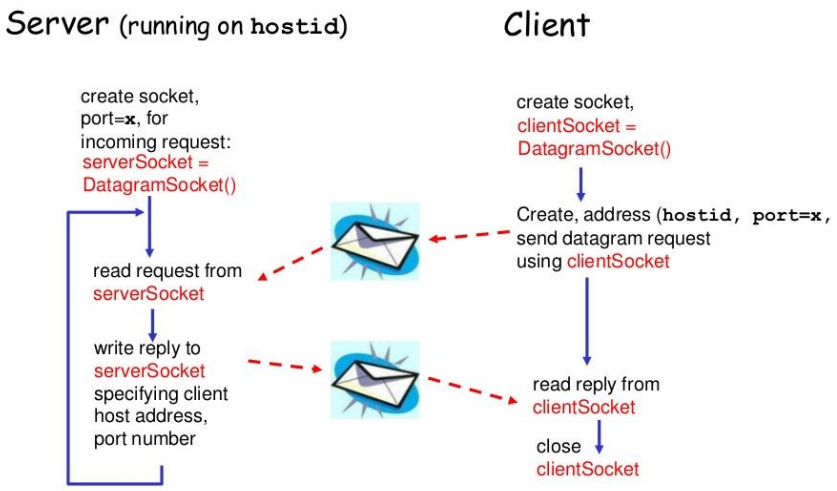


Figure: UDP client/server socket interaction

Use In Our project:

We have implemented UDP protocol in getRecordCount() method to extract the count of all the records from all 3 servers for 2 record types : DR Records and NR records. So whenever thread's start() method is called, run() method is called automatically by JVM:

```
public void run()
{
    DatagramSocket socket = null;

    try
    {
        socket = new DatagramSocket(this.UDPPort);
        byte[] message = new byte[1000];
        //Logger call

        while(true)
        {
            DatagramPacket request = new DatagramPacket(message, message.length);
            socket.receive(request);
            String data = new String(request.getData());
            String response = getRecordsFromServer(data);
            DatagramPacket reply = new DatagramPacket(response.getBytes(),response.length(),request.getAddress(),request.getPort());
            socket.send(reply);
        }
    }
    catch(Exception ex)
    {
        ex.printStackTrace();
    }
    finally
    {
        socket.close();
    }
}
```

6. TESTING

6.1 TESTING COVERAGE:

1) Following aspects have been tested on server side by creating sample manual tests:
   - Simple threading support
   - read and write concurrency
   - UDP communications
2) At least one of those 3 aspects is tested for each of the 4 functionality provided by the API:
   - create Doctor Record
   - create Nurse Record
   - get record counts
   - edit record

That means that all functionality have at least one test to ensure they're not broken.

6.2 TEST CLASSES AND METHODS:

Test case classes:

- ✓ DSMSConcurrencyTest.java
- ✓ DSMSFunctionalityTest.java

Test case methods under DSMSConcurrency.java :

- ✓ testServerAccess()

Test case methods under DSMSFunctionality.java :

- ✓ testServerAccess()
- ✓ testCreateDRecord()
- ✓ testCreateNRecord()
- ✓ testEditRecord()
- ✓ testGetRecordCount()

We have also created a Test Suite for all the classes to be executed at once:

- ✓ DSMSTestSuite.java

7. DIFFICULTIES

7.1 MAIN DIFFICULTIES: CONCURRENCY

The main difficulty of this assignment revolved around Multithreading and concurrency, particularly:

- How to implement FGL to allow different threads to access a shared data.
- How to avoid deadlocks in a readers-writers problem.
- How to start different servers from each individual threads.
- How to use Synchronization block for implementation of synchronization.
- Developing concurrency tests took some time (manipulating threads).

7.2 OTHER DIFFICULTIES:

- UDP implementation of getRecordcount function.

8. CONCLUSION

This assignment thus successfully demonstrates the use of Java RMI (Remote Method Invocation) to implement a DRMS with the help of UDP.

9. REFERENCES

HTTP://WWW.VOGELLA.COM/TUTORIALS/JAVACONCURRENCY/ARTICLE.HTML -- FOR CONCURRENCY
HTTP://WWW.NOESISPOINT.COM/JSP/SCJP/SCJPCH12.HTM -- FOR SYNCHRONIZATION
HTTP://WWW.SLIDESHARE.NET/DIFATTA/SE2-JA11-JAVANETWORKING42-AUDIO -- FOR UDP