COMP 6231

Distributed System Design

Assignment 2 Report

Distributed Class Management System (DCMS) using Java IDL

Submitted to: Professor Mohamed Taleb

By

Stallone Mecwan: 40161375

Jay Patel: 40163706

TABLE OF CONTENTS

Technique used

- We used **CORBA (Common Object Request Broker Architecture)** to implement the communication between Manager/Clients and servers (MTL, LVL and DDO).
- We used **UDP** to implement the communication between servers for counting Records on each server and transferring records across he servers.
- We used **HashMap** to store the records of teachers and students.
- We used **multithreading** technique so that multiple clients can act simultaneously.
- We used **synchronization and semaphore** technique to keep the integrity of data while modifying it, so the server can maximize the concurrency.
- The record class has been made to extend the Serializable class to implement the transfer record functionality as it allows writing of state of an object (here the record) into a byte-stream.

Design architecture

DCMS.idl

File that defines all the operations that can be used by the managers (Clients of this system).

- createTRecord
- createSRecord
- getRecordCounts
- editRecord
- displayAllRecords
- displayRecord
- transferRecord

When we compile this DCMS.idl with IDL Compiler, it generates 6 files: DCMSStub, DCMS, DCMSHelper, DCMSHolder, DCMSOperations and DCMSPOA.

MTL/LVL/DDOClass

These files extend the DCMSPOA. All the functions are implemented here. Some other functions are also defined here, i.e. validRecordID(), dateFormatChecker() (Date format checker for attribute statusDate of Student Record), setUpHashMap() (initialising HashMap here) and findRecord().

MTL/LVL/DDOServer

These files are responsible for getting the server up and running.

Server name, number and port are assigned here.

The Server file has the server's main() method, which:
• Creates and initializes an ORB instance

• Gets a reference to the root POA and activates the POAManager

• Creates a servant instance and tells the ORB about it

• Gets a CORBA object reference for a naming context in which to register the new CORBA object

• Gets the root naming context

• Registers the new object in the naming context

• Waits for invocations

As a starting point, some records are created here (by some default managerID like MTL/LVL/DDO0000) to fill the initial database. There are direct method invocations here, so we have also put validations in the MTL/LVL/DDOClass files.

MTL/LVL/DDOClient

These files contain client specific code. Here choices are provided to the user (manager) for which operation is to be performed. Validation of inputs is provided in these files.

• Creates and initializes an ORB

• Obtains a reference to the root naming context

• Looks up a string in the naming context and receives a reference to that CORBA object

• Invokes the object's methods and shutdown() operations and prints results

ManagerClient

This file is the starting point of the application. This file asks for the mangerID whose prefix (first 3 characters) are used to run the server. If the managerID is MTL1234, then MTL Server will be up and running and so on for other two servers. Here, validation of managerID takes place, if it is not in the format MTL/LVL/DDO followed by four digits, then it won't work.

Record

Record class implements Serializable class.It is used to create TeacherRecord and StudentRecord (subclasses). It has two main attributes of Record, RecordID and Name. Also, a toString() method is defined to appropriately format and print the data for the Client.

TeacherRecord

This class is used for storing various details of Teachers. It stores the following details of a student: First Name, Last Name, Address, Phone number, Specialization, Location. Similar to the Record class, Student Record class has a toStringT() method which formats and prints the data for the client.

StudentRecord

This class is used for storing various details of Students. It stores the following details of a student: First Name, Last Name, Courses he is registered in, status, status date. Similar to the Record class, Student Record class has a toStringS() method which formats and prints the data for the client.

Log

This file is used to create log files and add information of all activities taking place in the application.

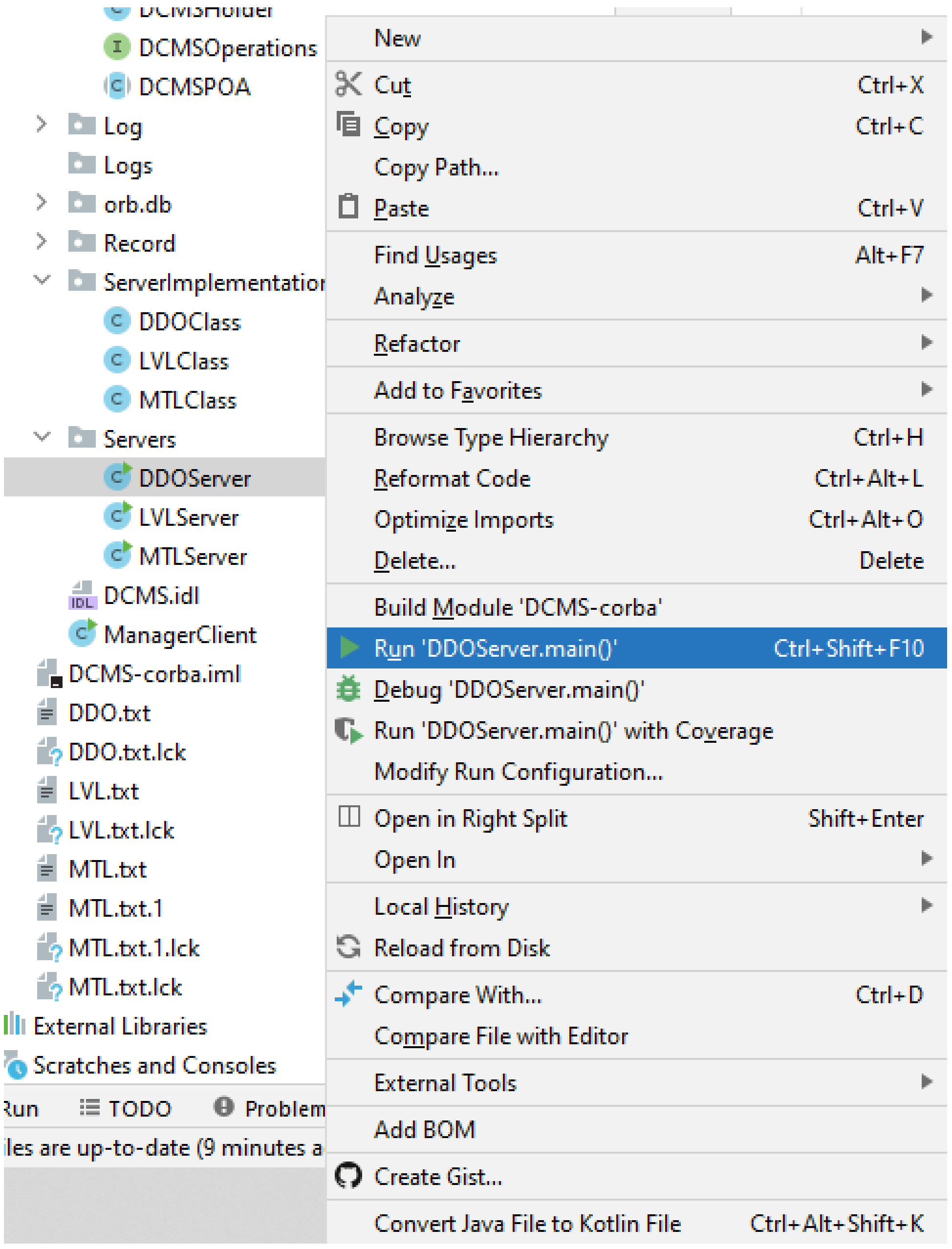
Server logs are named as MTL.txt, LVL.txt, DDO.txt are created in the project directory.
Client logs are saved as per the managerID used and are saved in the folder of “Logs” inside src.

Before starting the application

We have used Intellij IDEA Community edition (version 2020.3.2) and JDK 1.8 for development of this application.

Starting the application

1) cd src
2) start orbd -ORBInitialPort 1050
3) Start servers MTL, LVL and DDO (shown below) (all servers are needed to be running, for record count and transfer record functionality)
4) Start ManagerClient (similar to servers)



Run all servers and then ManagerClient in similar fashion (the command line arguments are set in external configuration of each file).

Rerun ManagerClient whenever a new client is to be used because we are using all three clients from a single file.

Data structures

Array List

All the details of students and teachers are stored in an array list

Hashmap

It consists of an id (string type) as the key and values are Array Lists that contain the records of students and teachers itself.

Test Scenarios

The ManagerClient is used to start the application. According to the ManagerID provided, the specific client object is created will run (prefix of the ID).

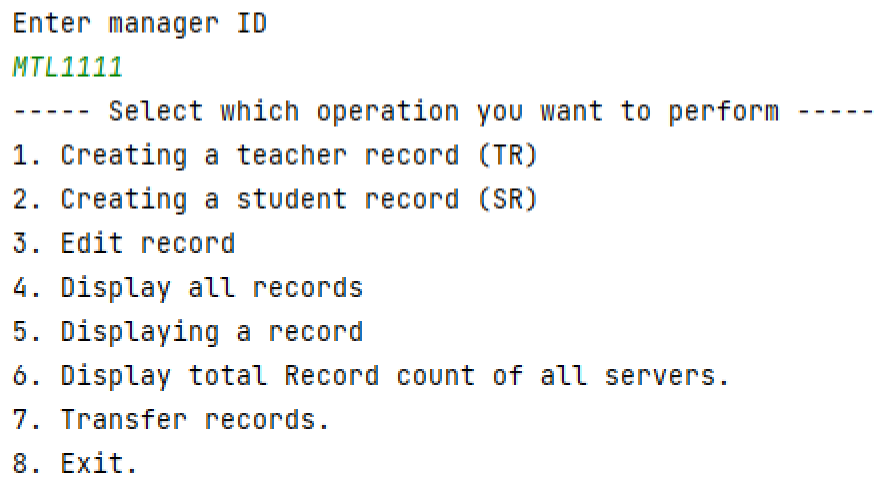
After this, the manager will be allowed to use the operations displayed to him in a menu based way.

For the entire application we have created functions that test several inputs alongwith the ones previously created in RMI implementation. Also, at many places we have put validations in the test cases itself at the point of input.

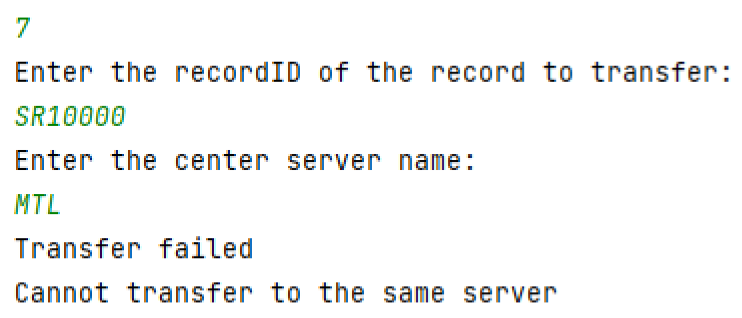
- validRecordID()
- hasNumbers()
- dateFormatChecker()
- hasAlpha()

These functions test the conditions on the input provided by the user to the system.

Here, we will be demonstrating a server trying to transfer record back at it again (self transfer) that would cause an error and that should not be accepted.

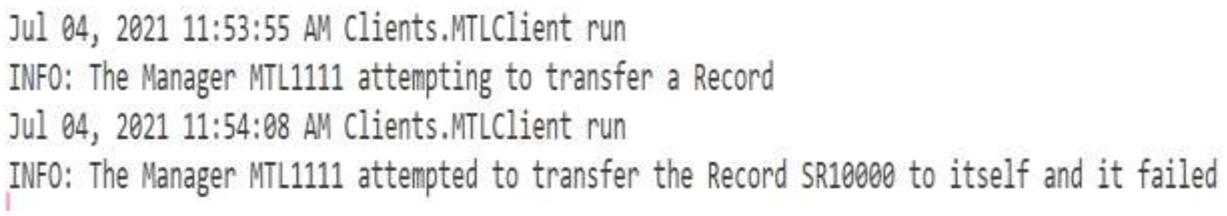


```
Enter manager ID
MTL1111
----- Select which operation you want to perform -----
1. Creating a teacher record (TR)
2. Creating a student record (SR)
3. Edit record
4. Display all records
5. Displaying a record
6. Display total Record count of all servers.
7. Transfer records.
8. Exit.
```



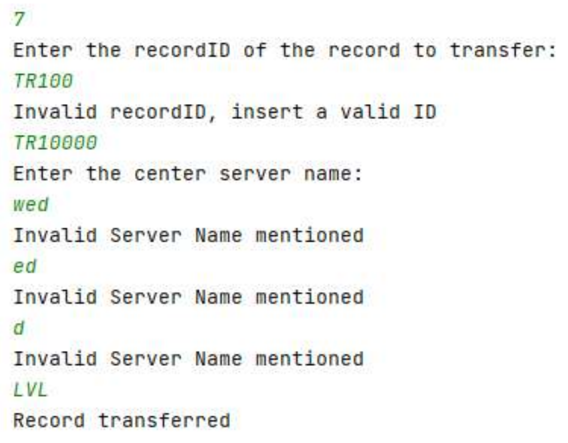
```
7
Enter the recordID of the record to transfer:
SR10000
Enter the center server name:
MTL
Transfer failed
Cannot transfer to the same server
```

Below is the screenshot of MTL1111 log file



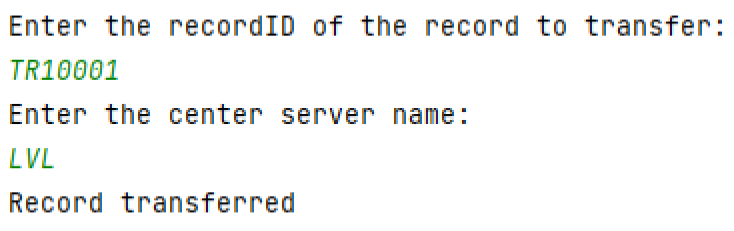
```
Jul 04, 2021 11:53:55 AM Clients.MTLClient run
INFO: The Manager MTL1111 attempting to transfer a Record
Jul 04, 2021 11:54:08 AM Clients.MTLClient run
INFO: The Manager MTL1111 attempted to transfer the Record SR10000 to itself and it failed
```

If we enter an invalid Server name, then it would not work and ask for a valid server name

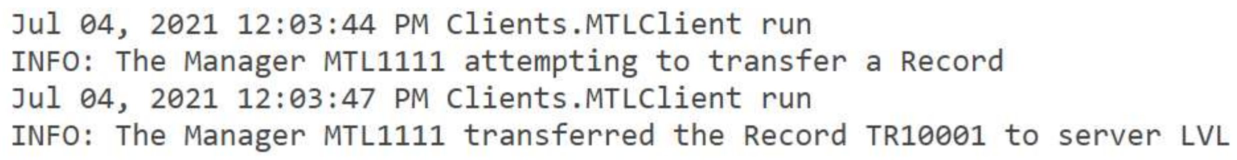


```
7
Enter the recordID of the record to transfer:
TR100
Invalid recordID, insert a valid ID
TR10000
Enter the center server name:
wed
Invalid Server Name mentioned
ed
Invalid Server Name mentioned
d
Invalid Server Name mentioned
LVL
Record transferred
```

If a proper server name is inserted, then the record gets successfully transferred.

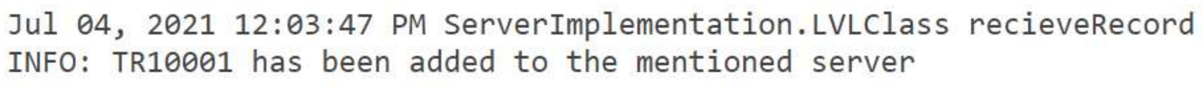


```
Enter the recordID of the record to transfer:
TR10001
Enter the center server name:
LVL
Record transferred
```



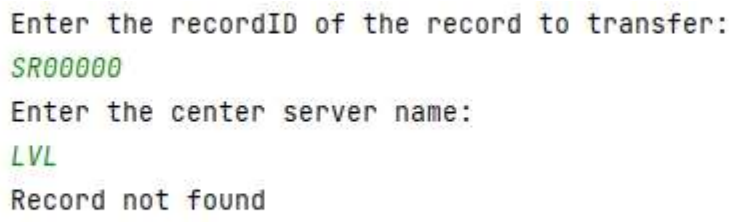
```
Jul 04, 2021 12:03:44 PM Clients.MTLClient run
INFO: The Manager MTL1111 attempting to transfer a Record
Jul 04, 2021 12:03:47 PM Clients.MTLClient run
INFO: The Manager MTL1111 transferred the Record TR10001 to server LVL
```

This is the screenshot of LVLServer log file.



```
Jul 04, 2021 12:03:47 PM ServerImplementation.LVLClass recieveRecord
INFO: TR10001 has been added to the mentioned server
```

If we try to transfer a record which is not present in the server, then it would give a record not found message.



```
Enter the recordID of the record to transfer:
SR00000
Enter the center server name:
LVL
Record not found
```

Important part/ Difficulty

The RMI implementation of the assignment 1 included running the servers and clients from a single file.

This thing could not be achieved with the CORBA implementation. At any point when the server started running, the code did not return the control to the client program as it did in the RMI build.

This was the challenging part, which was overcame by separating the servers and running them explicitly.

The client usage is administered by the ManagerClient.