COMP 6231
Distributed Systems Design

Web Service Implementation of Distributed Class Management System (DCMS)

Submitted To: Prof. M. Taleb

Contributors

Zalakben Rajendrakumar Patel (40164315)

Manimaran Palani (40167543)

Description:

Distributed Class Management system (DCMS) is implemented as a distributed system with the use of web service to be used by centre managers, who are responsible for managing teachers' and students' information across 3 centres: Montreal, Laval and Dollard-des-Ormeaux. Each location has its own server and maintains teachers' and students' records. Centre managers are responsible for adding, editing teachers' and students' records at his/her own centre. Additionally, manager can also get count of records present at other servers through UDP connection. Along with these features, managers can transfer record of their location to another location. This system is built by using Java web services where manager can see a single system performing various simultaneous requests with proper synchronization.

Design Architecture:

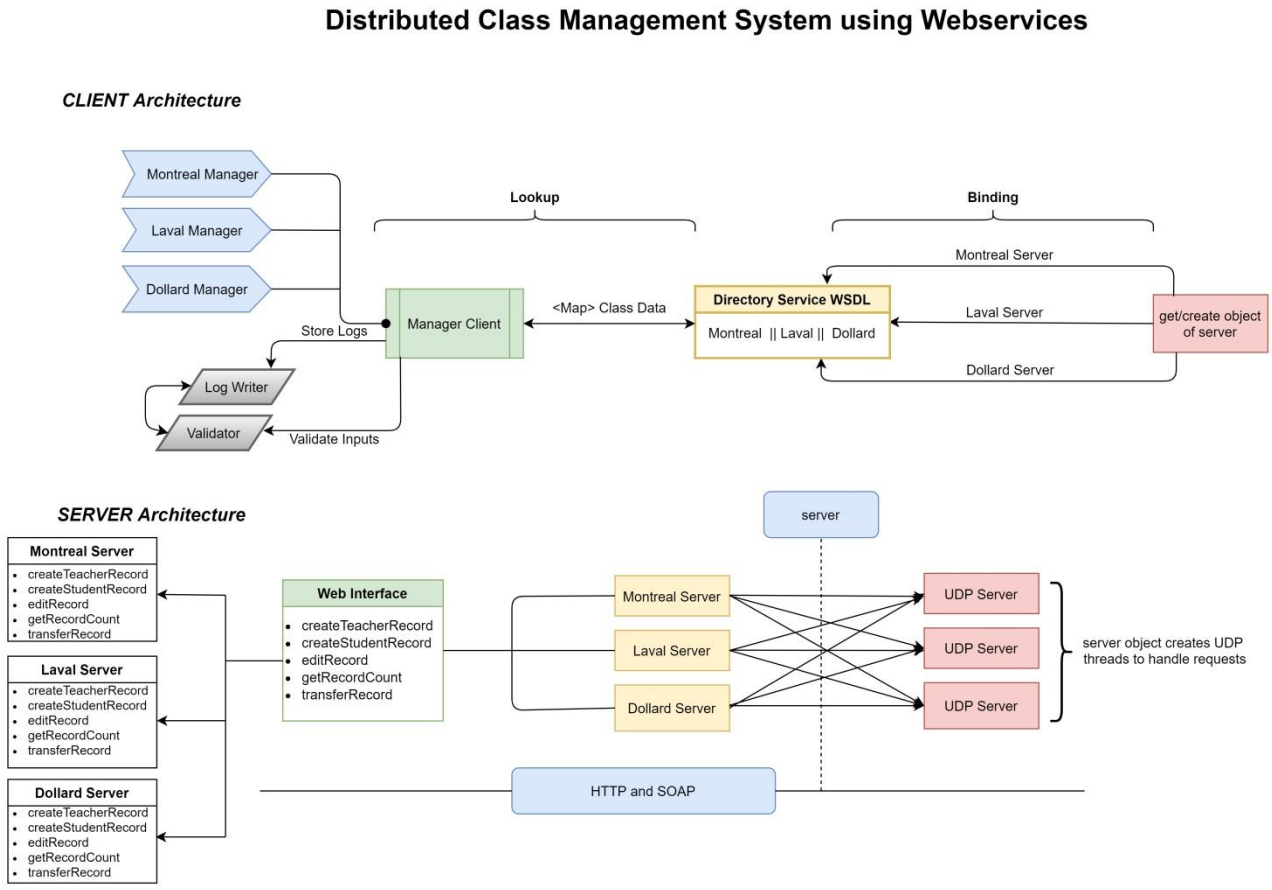

Figure 1 – Architectural Design

Architecture Explanation:

Distributed Class Management system (DCMS) contains 3 Servers named MontrealServer (Port 8880), LavalServer (Port 8881) and DollardServer (Port 8882). Each of these servers is interconnected with other two by UDP connections. Class MainServer is responsible for establishing server connections and publishing server instances to certain endpoints. ManagerClient class is client responsible for accessing the system through manager logins, which invokes required location's server end point in order to perform operation. MontrealServer, LavalServer and DollardServer provide the implementation of methods defined in java interface: DCMSInterface.java and it manipulates ConcurrentHashMap defined at individual servers based on operations triggered by managers.

Class Diagram:

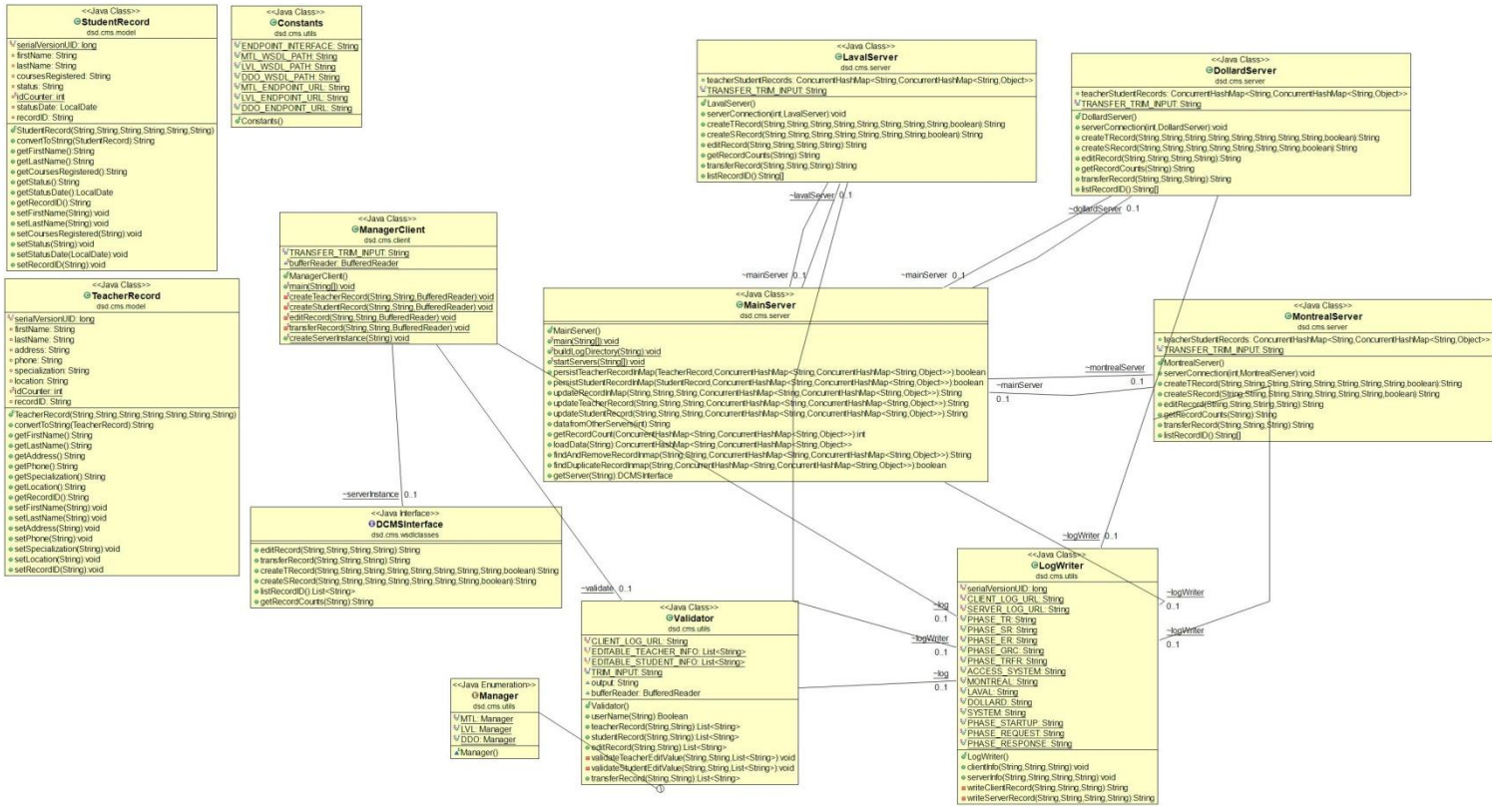

Figure 2 – DCMS Class Diagram

1) DCMSInterface (Web Service Interface):

Interface annotated with @WebService. Responsible for giving abstract declaration of functionalities like, adding Teacher/Student record, editing existing record, getting count of records from all servers, transferring record from one server to other server as below:

- string createTRecord(in string recordId, in string firstName, in string lastName, in string address, in string phone, in string specialization, in string location, in string managerID, in boolean transferStatus);
- string createSRecord(in string recordId, in string firstName, in string lastName, in string courseRegistered, in string status, in string statusDate, in string managerID, in boolean transferStatus);
- string getRecordCounts(in string managerID);
- string editRecord(in string recordID, in string fieldName, in string newValue, in string managerID);
- listRecordID listRecordID();
- string findRecord(in string recordID);

2) MontrealServer, LavalServer, DollardServer (Web Service Implementation Servers):

These server classes annotated with @WebService provide actual implementation for functionalities defined in web service Interface (DCMSInterface.java). These servers are being invoked from manager client based on the manager's location. Servers are also responsible for publishing their end points and receiving UDP data packets from other servers for getting record count.

3) MontrealServerService, LavalServerService, DollardServerService:

These are auto generated classes by executing **wsgen** for creating wsdl and **wsimport** commands for importing wsdl. These classes are mapping of web service implementations from the wsdl files. Manager client can find required servers by invoking server port methods defined in these service classes.

3) MainServer:

MainServer is a main class which is responsible for starting all three locations' servers by invoking their 'serverConnection' method, which further publishes/initiates endpoint of different servers and establishes udp connection. It also has few methods which are common across all the servers in terms of implementation and which avoids the duplication of code. It binds each servers object to a specific endpoint.

- Endpoint dollardServerEndPoint = Endpoint.publish(Constants.DDO_ENDPOINT_URL, dollardServer);
- Endpoint montrealServerEndPoint = Endpoint.publish(Constants.MTL_ENDPOINT_URL, montrealServer);
- Endpoint lavalServerEndPoint = Endpoint.publish(Constants.LVL_ENDPOINT_URL, lavalServer);

4) ManagerClient:

This client class is responsible for taking inputs from user about identifying manager ID, operations which are to be performed at that location and all required inputs for performing that operation. ManagerClient looks up the naming context reference in order to find correct location sever and invokes required method.

7) StudentRecord, TeacherRecord (Models):

StudentRecord class is a data model class which has fields like first name, last name, courses registered, status (Active/InActive), status date and record id which are responsible for persisting new student's data. TeacherRecord class is a data model class which has fields like first name, last name, address, phone, specialization, location and record id which are responsible for persisting new teacher's data at any location.

9) Regex & Validator:

Regex class is responsible for performing the pattern matching according to defined regex patterns for inputs of type string, number, date etc. While validator class performs all required validations on the given inputs by manager before redirecting to actual implementation of servers. It requires first name, last name to be string, phone number to be a number, location from specific list: MTL, LVL, DDO, status date to be in specific date format etc.

Data Structure/Model:

ConcurrentHashmap is used as a data model, which is maintained at each server locations. Key of this map is an alphabet whereas value against that is a child ConcurrentHashmap of Teacher/Student record details whose last name starts with that alphabet. Key of this child map is record id and value of the child map is the record object.

Ie. `ConcurrentHashMap<String, ConcurrentHashMap<String, Object>>`

Example: If we have student record with first name as 'James', it will create/update ConcurrentHashMap to have this student's information against key 'J'. Value of this ConcurrentHashMap will contain child ConcurrentHashMap as a value, key of which would be 'SR10000'(Auto-created record id for given student), and value would be instance/object of StudentRecord. Similarly if Teacher record is created, child map's key will be teacher id (TR*****).

The Most Important/Difficult Part:

- Handling multiple requests coming at the same or different locations concurrently, blocking resources for proper updates of the data, implementing concurrency and synchronization have been key aspect and challenging part of the assignment. For this, we have used keyword synchronized in order to apply lock on shared variable such as ConcurrentHashMap. ConcurrentHashMap is comparatively more thread safe and achieves required concurrency. For example, application should not fail when different managers tries to manipulate the same record at the same time, one editing the record, one transferring the record.
- Restructuring code in order to remove corba related files and changing code for webservice implementation.

Logging:

To keep track of all the operations performed by manager and all the operations performed at particular location, separate logs files are being maintained for both managers and locations.

Concurrency & Synchronization:

New thread is created to communicate to each of the servers in order to handle multiple add/edit/get record count/transfer record requests coming at the same time at different locations. In order to handle multiple requests in single location, coming at the same time, synchronized methods are used, which can block its implementation to have only one thread executing inside them. With this synchronization technique, we can avoid multiple requests trying to update the same ConcurrentHashMap at the same time which can lead to faulty results.

UDP Server Connection:

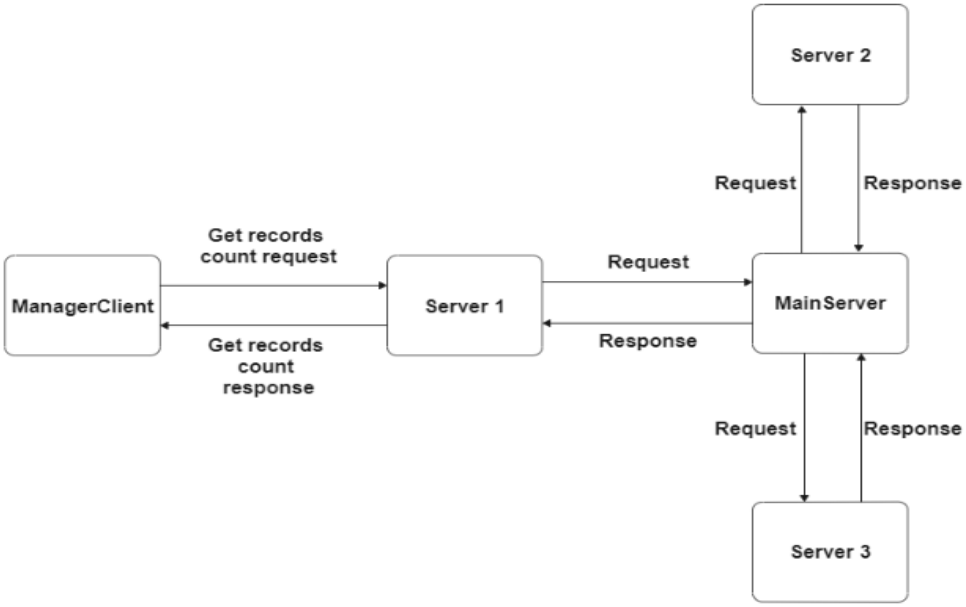

Figure 3 – UDP Server Design

Initial Data Loading:

All three location servers will be loaded initially with sample data stored in text files: MontrealData, DollardData, LavalData located at src/main/resources.

Client - Server Execution Flow:

Below are the steps performed in sequence when any manager tries to log in and performs any operation.

- All 3 location based servers have to be started by running MainServer, which further publishes server instances at required webservice endpoints.
- ManagerClient performs required validations of input information passed by manager. Post validation success, ManagerClient communicates with either of three servers based on correct instance returned by calling server port methods.
- Location server receives the data and performs the requested operation. If required, it also invokes other locations' server to fetch the data through UDP connection. After performing operation, location server will return back the success/failure/information results to the ManagerClient.

Run the Application:

- Run MainServer.java which will start all three location based servers.
- Run ManagerClient.java to execute required operations.
- Run TestManagerClient.java in order to verify basic and advanced level test cases.

Test Scenarios:

1) Manager ID given in incorrect format :
(System will give error if manager ID given is not in form of MTL**** or LVL**** or DDO****.

```
Distributed Class Management System
===================================

Create an User to interact with the system
 Format: MTL**** or LVL**** or DDO****
         *****:numbers

Enter an User Name:
XYZ1111
===== user name is invalid =====
```

2) New teacher creation test scenarios :

a. Validation failure in creating teacher with location other than MTL / LVL / DDO :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
1
Enter teacher record inputs in the following format

FirstName, LastName, Address, Phone, Specialization(Format: JAVA|HTML5|PHP), location (Format : MTL or LVL or DDO)
Example: Kevin, Peterson, 23,Lords street,London-987, 9023123445, Java|Python|Scala, MTL

enter a teacher record :
Kevin, Peterson, 23,Lords street,London-987, 9023123445, Java|Python|Scala, TRO

Location is invalid
Please match the format mentioned above.
```

b. Successfully creating new teacher record when all validations are passed:

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
1
Enter teacher record inputs in the following format

FirstName, LastName, Address, Phone, Specialization(Format: JAVA|HTML5|PHP), location (Format : MTL or LVL or DDO)
Example: Kevin, Peterson, 23,Lords street,London-987, 9023123445, Java|Python|Scala, MTL

enter a teacher record :
Kevin, Peterson, 23,Lords street,London-987, 9023123445, Java|Python|Scala, MTL

FirstName , LastName  Address , phone , specialization , location
[Kevin, Peterson, 23,Lords street,London-987, 9023123445, Java,Python,Scala, MTL]

Press S to save the current record or Press C to enter new record :
S
Teacher record TR10000 created in Montreal location with name : Kevin Peterson
```

c. Similarly there are validation scenarios for accepted format of first name, last name, phone number attributes of teacher record creation.

3) New student creation test scenarios :

a. Validation failure in creating student with incorrect status date format :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
2
Enter Student record inputs in the following format

FirstName, LastName, CourseRegistered(Format: JAVA|HTML5|PHP), Status(Format: 1-Active or 0-InActive), StatusDate(Format: yyyy-mm-dd)
Example: James, Anderson, Java|Python|Scala, 1, 2021-12-22

enter a Student record :
James, Anderson, Java|Python|Scala, 1, 2021-12-2222
```

b. Successfully creating new student record when all validations are passed :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
2
Enter Student record inputs in the following format

FirstName, LastName, CourseRegistered(Format: JAVA|HTML5|PHP), Status(Format: 1-Active or 0-InActive), StatusDate(Format: yyyy-mm-dd)
Example: James, Anderson, Java|Python|Scala, 1, 2021-12-22

enter a Student record :
James, Anderson, Java|Python|Scala, 1, 2021-12-22

firstName, lastName, courseRegistered, status, statusDate)
[James, Anderson, Java,Python,Scala, Active, 2021-12-22]

Press S to save the current record or Press C to enter new record :
S
Student record SR10000 created in Montreal location with name : James Anderson
```

c. Similarly validations for acceptable string for first name, last name, status for student record.

4) Edit existing teacher/student record :

a. Editing teacher/student record with invalid Record ID (Invalid record ID format or record ID which doesn't exists in particular server) :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
3

Enter input to edit a record in the following format

RecordId, fieldName, newValue

Teacher Record Format: ("TR*****", "ADDRESS or PHONE or LOCATION", "newValue")
Example : (TR12345, PHONE, 9023412231)

Student Record Format: ("SR*****", "COURSEREGISTERED or STATUS or STATUSDATE", "newValue")
Example: (SR12345, COURSEREGISTERED, JAVA|HTML5|PHP)

enter a record to edit :
TR12345, PHONE, 9023412231

(RecordId, fieldName, newValue)
[TR12345, PHONE, 9023412231]

 Press E to proceed with editing or Press C to enter new record to edit :
E
Teacher Record with given id : TR12345 was not found.
```

b. Editing teacher/student record with invalid field name (Editing field which isn't allowed to be modified will result to a validation failure) :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
3

Enter input to edit a record in the following format

RecordId, fieldName, newValue

Teacher Record Format: ("TR*****", "ADDRESS or PHONE or LOCATION", "newValue")
Example : (TR12345, PHONE, 9023412231)

Student Record Format: ("SR*****", "COURSEREGISTERED or STATUS or STATUSDATE", "newValue")
Example: (SR12345, COURSEREGISTERED, JAVA|HTML5|PHP)

enter a record to edit :
TR10000, XYZ, 9023412231
Field Name is incorrect. Please match the format mentioned above.
```

c. Editing teacher/student record with incorrect new value (Example : Invalid value for location) :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
3

Enter input to edit a record in the following format

RecordId, fieldName, newValue

Teacher Record Format: ("TR*****", "ADDRESS or PHONE or LOCATION", "newValue")
Example : (TR12345, PHONE, 9023412231)

Student Record Format: ("SR*****", "COURSEREGISTERED or STATUS or STATUSDATE", "newValue")
Example: (SR12345, COURSEREGISTERED, JAVA|HTML5|PHP)

enter a record to edit :
TR10000, LOCATION, TRO
Value is in incorrect. Please match the format mentioned above.
```

d. Successfully editing teacher/student record when all validations are passed :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
3

Enter input to edit a record in the following format

RecordId, fieldName, newValue

Teacher Record Format: ("TR*****", "ADDRESS or PHONE or LOCATION", "newValue")
Example : (TR12345, PHONE, 9023412231)

Student Record Format: ("SR*****", "COURSEREGISTERED or STATUS or STATUSDATE", "newValue")
Example: (SR12345, COURSEREGISTERED, JAVA|HTML5|PHP)

enter a record to edit :
SR10000, COURSEREGISTERED, JAVA|HTML5|ML

(RecordId, fieldName, newValue)
[SR10000, COURSEREGISTERED, JAVA,HTML5,ML]

 Press E to proceed with editing or Press C to enter new record to edit :
E
Edit operation performed successfully.
Updated Record: Id:SR10000, FN:James, LN:Anderson, CoursesRegistered:JAVA,HTML5,ML, Status:Active, StatusDate:2021-12-22
```

5) Finding number of records present at each server locations :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
4
MTL : 9, LVL : 6, DDO : 8
```

6) Transferring teacher/student record to another server location :

a. Trying to transfer invalid record which doesn't exists in particular location :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
5
Enter inputs in the following format to Transfer Record

recordID (Format : TR9999/SR9999), remoteCenterServerName(Format : MTL or LVL or DDO)
Example: TR9999, LVL

Available records to transfer: TR10000,SR10000,SR99991,SR99992,SR99993,TR99994,TR99992,TR99993,TR99991

enter details to transfer record :
TR10001, LVL
TR10001 Not Found to transfer
```

b. Transfer record successful from Montreal to Laval :

```
Choose any of the options below to access the system
1 : Create a Teacher Record
2 : Create a Student Record
3 : Edit a Record
4 : Get Record Count
5 : Transfer Record
C : To create new user
Select :
5
Enter inputs in the following format to Transfer Record

recordID (Format : TR9999/SR9999), remoteCenterServerName(Format : MTL or LVL or DDO)
Example: TR9999, LVL

Available records to transfer: TR10000,SR10000,SR99991,SR99992,SR99993,TR99994,TR99992,TR99993,TR99991

enter details to transfer record :
TR10000, LVL
Teacher record TR10000 transferred to Laval location with name : Kevin Peterson
```

7) Basic and advanced concurrency access (Two managers trying to manipulate same record) scenarios :

a) Basic Scenarios result through TestManagerClient.java

- Servers are started with initial data
- Checking correctness of creating teacher and student record by auto log-in of Montreal manager.
- Verifying edit record functionality on newly created student record.
- Verifying record count from Montreal manager log-in.
- Transferring newly created teacher record from Montreal to Laval.
- Verifying record count from Laval manager log-in.
- Checks edit record correctness on newly transferred teacher record at Laval location.

```
<terminated> TestManagerClient (1) [Java Application] C:\Program Files\Java\jdk1.8.0_291\bin\javaw.exe  (18-Jul-2021, 5:35:49 pm – 5:36:03 pm)
****************** Basic Test Cases ******************


Test 1 : Create teacher record on Montreal Server
Executing : createTeacherRecord() , Teacher record TR10000 created in Montreal location with name : testFName1 testLName1

Test 2 : Create student record on Montreal Server
Executing : createStudentRecord() , Student record SR10000 created in Montreal location with name : testFName1 testLName1

Test 3 : Edit existing record on Montreal Server
Executing : editRecord() , Edit operation performed successfully.
Updated Record: Id:SR10000, FN:testFName1, LN:testLName1, CoursesRegistered:JAVA|HTML5, Status:Active, StatusDate:2021-12-24

Test 4 : Get Record Count from Montreal location
Executing : getRecordCount() , MTL : 9, LVL : 6, DDO : 8

Test 5 : Transfer teacher record to Laval
Executing : transferRecord() , RECORDTOTRANSFER Not Found to transfer

Test 6 : Get Record Count from Laval location
Executing : getRecordCount() , MTL : 9, LVL : 6, DDO : 8

Test 7 : Edit transferred record on Laval Server
Executing : editRecord2() , Student Record with given id : SR10000 was not found.
```

b) Advanced (Concurrency) Scenarios result through TestManagerClient.java

- Initially 1 Teacher record with first name: 'fEditTransfer1' and 1 Student record with first name: 'fEditTransfer2' will be created in Montreal location through Montreal manager auto log-in.
- We have made 2 threads which are trying to attempt edit record and transfer record on these 2 records at almost same time.
- Thread 1 tries to edit teacher record at Montreal location.
- Thread 2 tries to transfer this teacher record to Laval.
- Thread 1 tries to edit student record at Montreal location.
- Thread 2 tries to transfer this student record to Laval.
- Application will run without any error. When edit and transfer both are attempted on single record, only 1 of them will be executed.

```
****************** Concurrency Test Cases ******************


** Initially 1 Teacher record with first name : fEditTransfer1 and 1 Student record with first name : fEditTransfer2 will be created **
** There are 2 threads trying to attempt edit record and transfer record on these 2 records **
** Thread 1 tries to edit teacher record and transfer student record to laval **
** Thread 2 tries to transfer teacher record and edit student record to laval **
** Application will run without any error. When edit and transfer both are attempted on single record, only 1 of them will be executed. **

Thread 1 Method : createPlayerAccount() ,  First Name : fEditTransfer1 , Teacher record TR10001 created in Montreal location with name : fEditTransfer1 lEdi

Thread 1 Method : createPlayerAccount() ,  First Name : fEditTransfer2 , Student record SR10001 created in Montreal location with name : fEditTransfer1 lEdi

Thread 16 Executing Transfer Record on TR10001, First Name : fEditTransfer1

Thread 15 Executing Edit Record on TR10001, First Name : fEditTransfer1

Thread 15 Edit Record Result for TR10001:: Edit operation performed successfully.
Updated Record: Id:TR10001, FN:fEditTransfer1, LN:lEditTransfer1, Address:abc-12, Montreal, Phone:9191919191, Specialization:Python|Java, Location:MTL

Thread 15 Executing Transfer Record on SR10001, First Name : fEditTransfer2

Thread 16 Transfer Record Result for TR10001 :: Teacher record TR10001 transferred to Laval location with name : fEditTransfer1 lEditTransfer1

Thread 16 Executing Edit Record on SR10001, First Name : fEditTransfer2

Thread 15 Transfer Record Result for SR10001 :: Student record SR10001 transferred to Laval location with name : fEditTransfer1 lEditTransfer1

Thread 16 Edit Record Result for SR10001 :: Student Record with given id : SR10001 was not found.
```