**COMP 6231 Assignment 2 Report**

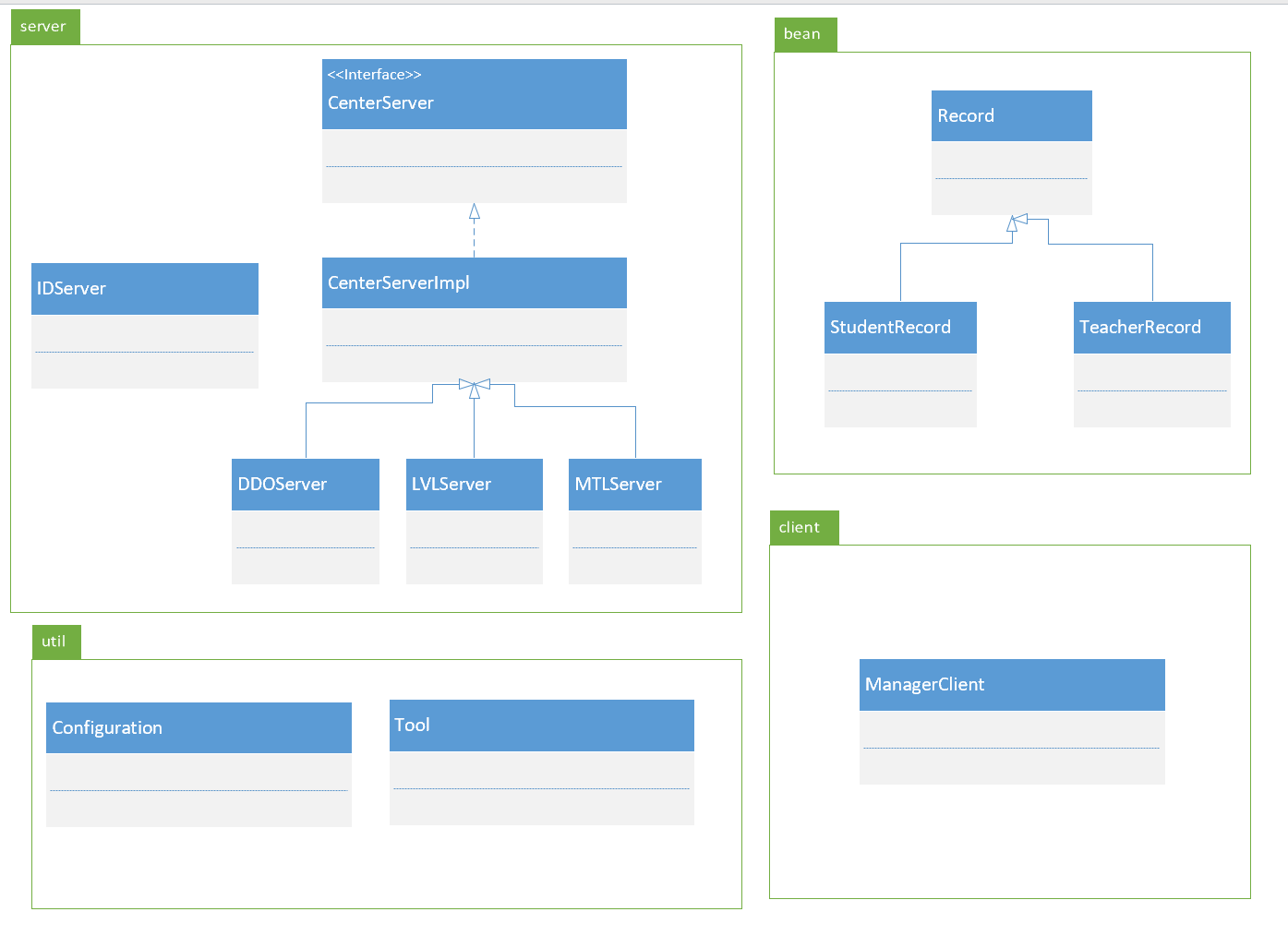
**Group Member:**

**Zexin Peng**

**Student number: 40166520**

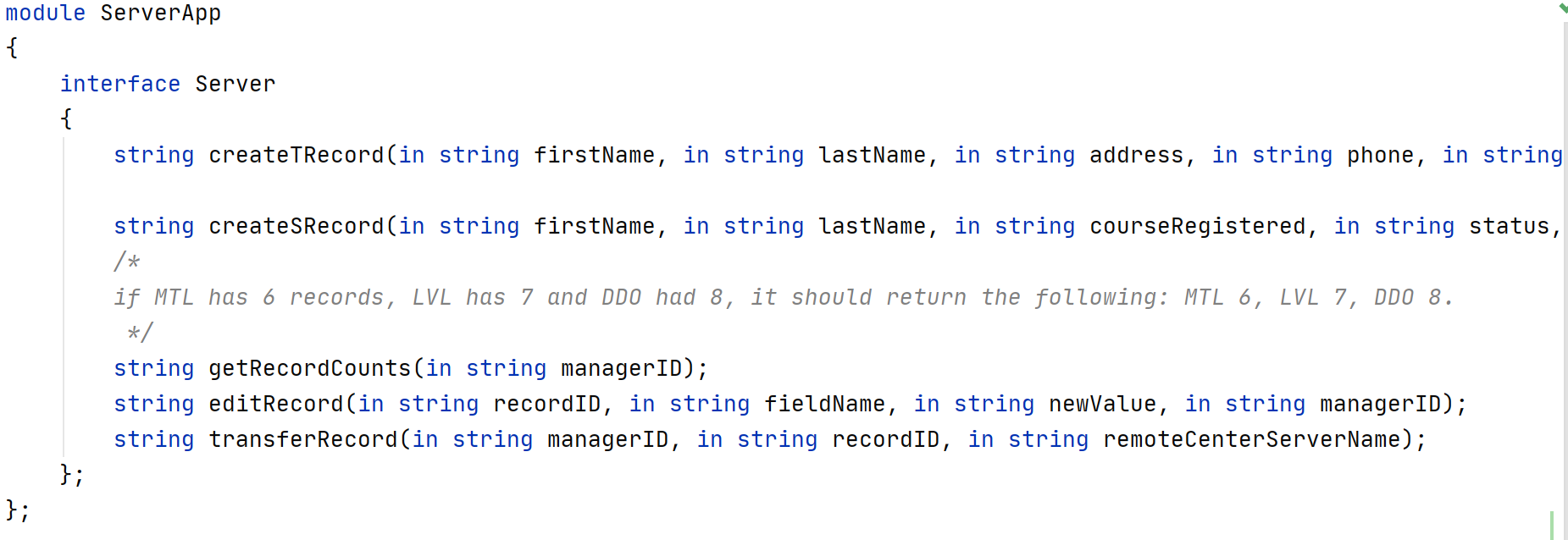
1. **Architecture Design:**

DDOServer, LVLServer and MTLServer are inherited from the same superclass “CenterServerImpl”, which implements the Interface “CenterServer”. IDServer is responsible for assigning RecordID when creating new records. We should indicate the managerID and location of the client at first. The responsibility of “Configuration” is to store some configuration information like ports and directory names. “Tool” provides basic support to other classes.

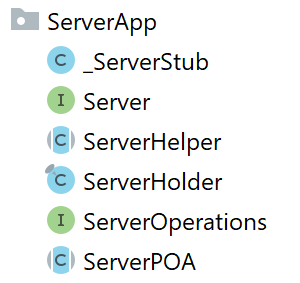


1. **Techniques：**
   1. **CORBA:**

We need to define the interface in IDL language at first.



And then, we need to compile it and generate the according JAVA code.



* 1. **UDP/TCP Socket:**

I use UDP socket to get the quantity of records in other servers.

The recordID should be unique in the system, so I create a new server named “IDServer” to assign recordID, and it uses TCP to communicate with other servers. Another usage of TCP programming in my assignment is in method transferRecord().

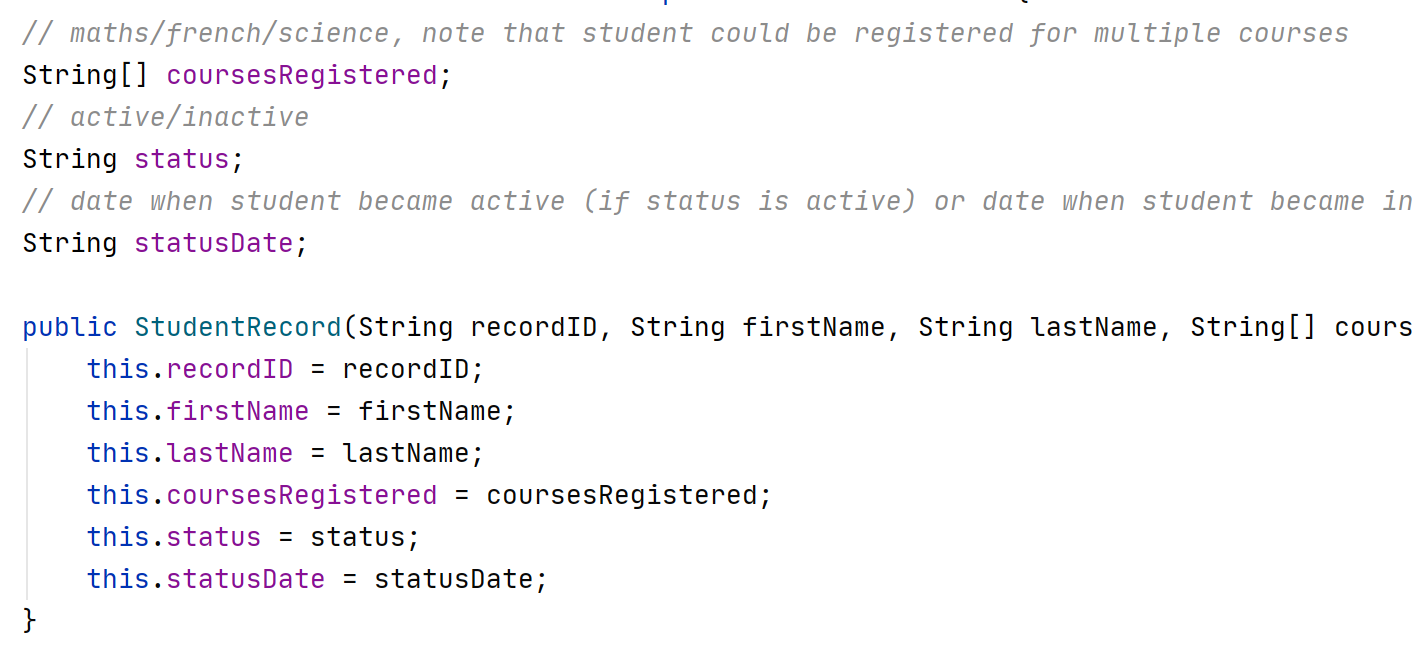
* 1. **synchronized:**

I use ‘synchronized’ in my code to avoid synchronization problem, because some shared data cannot be modified concurrently. By the way, to maximize concurrency, I do not maintain strong consistency in my assignment. In other word, if a server has 10 records, and a client will get 10 and will not be blocked when it wants to know the quantity of records in this server even if another client is creating new records in this server at the same time. If a system has strong consistency, the client will be blocked and get 11 after another client creates a new record, but it will reduce the concurrency.

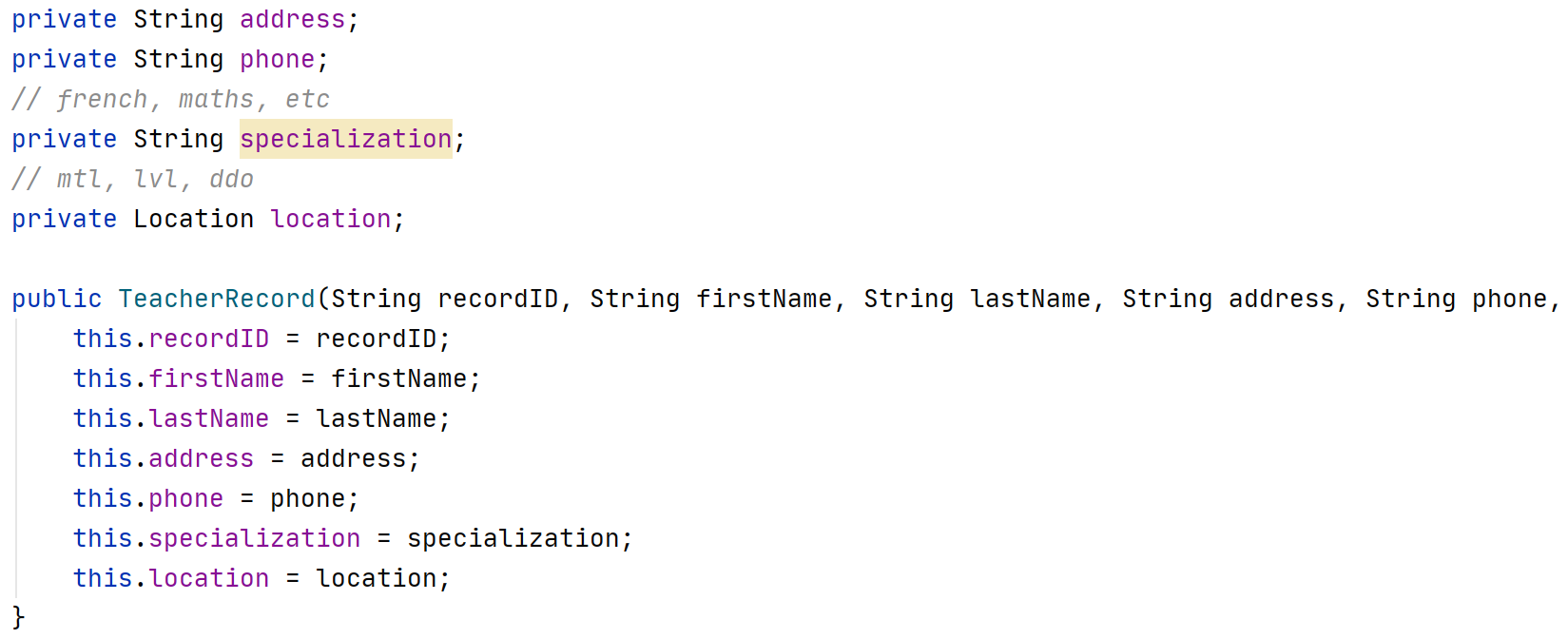
1. **Data Structure:**

I encapsulate student records and teacher records as Bean classes with bunches of getter and setter method.

* 1. **StudentRecord:**



* 1. **TeacherRecord:**



* 1. **RecordMap:**

All records are stored in the RecordMap, whose type definition is HashMap<Character, List<Record>>. The key of this map is the first letter of the last name indicated in the records.



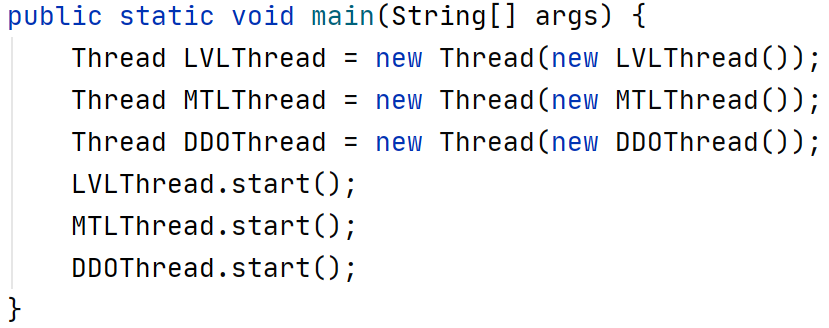
1. **Test scenarios**

each of the servers initially has a few records except scenario 4.1. We have two records in MTL Server, three records in LVL Server and four records in DDO Server. The correctness of methods createTeacherRecord(), createStudentRecord(), getRecordCounts() and editRecord() are already tested in the report of assignment 1, so I focus on concurrency of my system in this report. All codes can be found in package client.

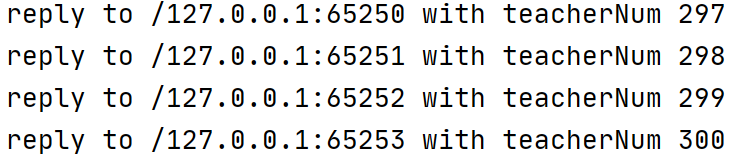
* 1. **Scenario 1:**

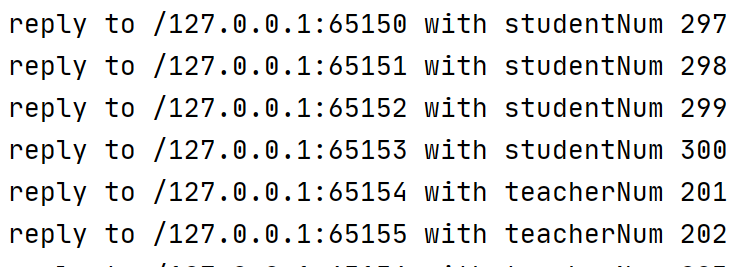
**Description:** I start three threads representing three clients “LVL0001”, “DDO0001” and “MTL0001” respectively and create 300 student records and 300 teacher records concurrently to test the uniqueness of recordID and correctness of createRecords. You can see the code in client.TestScenario1.java.

Firstly, I create three threads and run them.

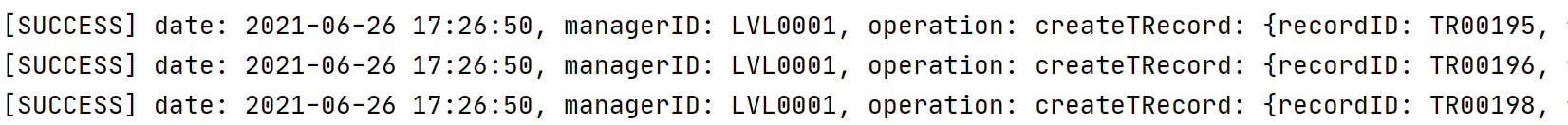


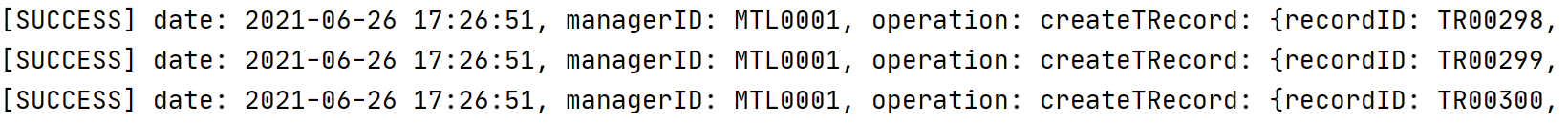
In the IDServer, we got logs below.

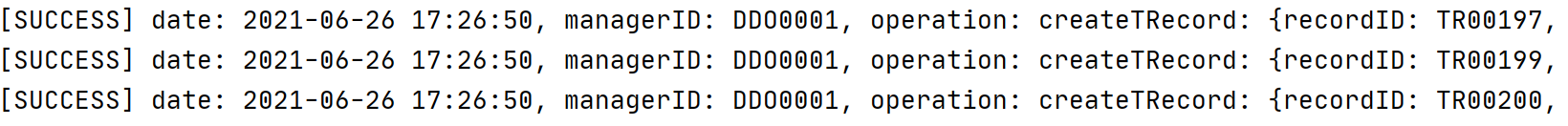




Logs in three servers are below.







At last, we invoke the method getRecordCounts(), we got the result as below. The last student recordID is SR00300, and the last teacher recordID is TR00300. After analysis, we got a conclusion that all recordIDs are unique in the system.



* 1. **Scenario 2:**

**Description:** this test scenario focus on transferRecord() method in the server. I will edit all field in student record and teacher record by three managers “LVL0001”, “MTL0001” and “DDO0001”. By this test scenario, we will know the correctness of editRecordCounts method and log function.

Firstly, we will try some invalid input.

1. **The recordID that does not exist. We try it in LVL client.**



We will get log message as below, and messages are stored in the log files LVL0001 and LVL.



1. **We try the invalid location name in MTL client.**



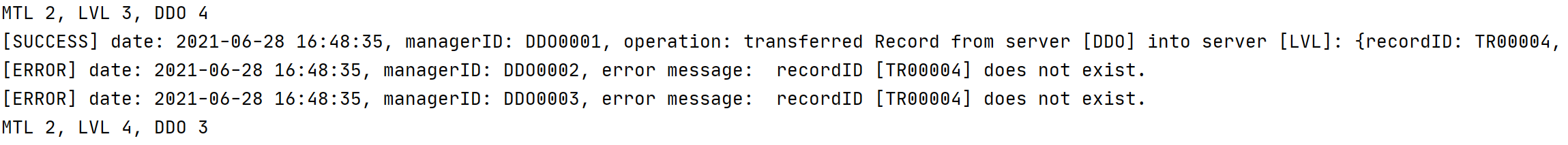
We will get log message as below, and messages are stored in the log files MTL0001 and MTL. 

Secondly, we will test the correctness of the method, respectively are atomicity and concurrency.

1. **We test the atomicity of the method in DDO Server. We start three threads with client manager “DDO0001”, “DDO0002” and “DDO0003”, and the record can only be transferred once.**



We got log messages as below, and the quantity of records in the server is also consistent.



We can see that the record “TR0004” is only transferred once. Then we will see the correctness of log function. We get three new log files.



Contents in DDO0001, DDO0002 and DDO003 are showed in order as below, and the log messages are consistent with that in the client console.



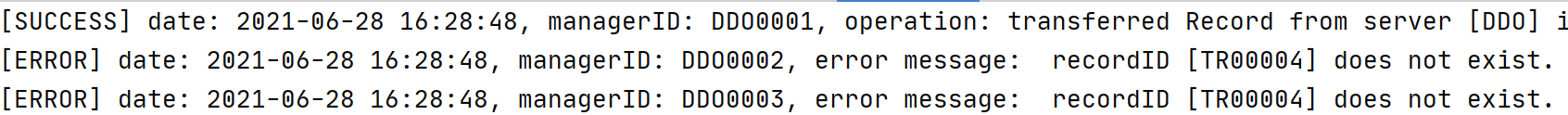




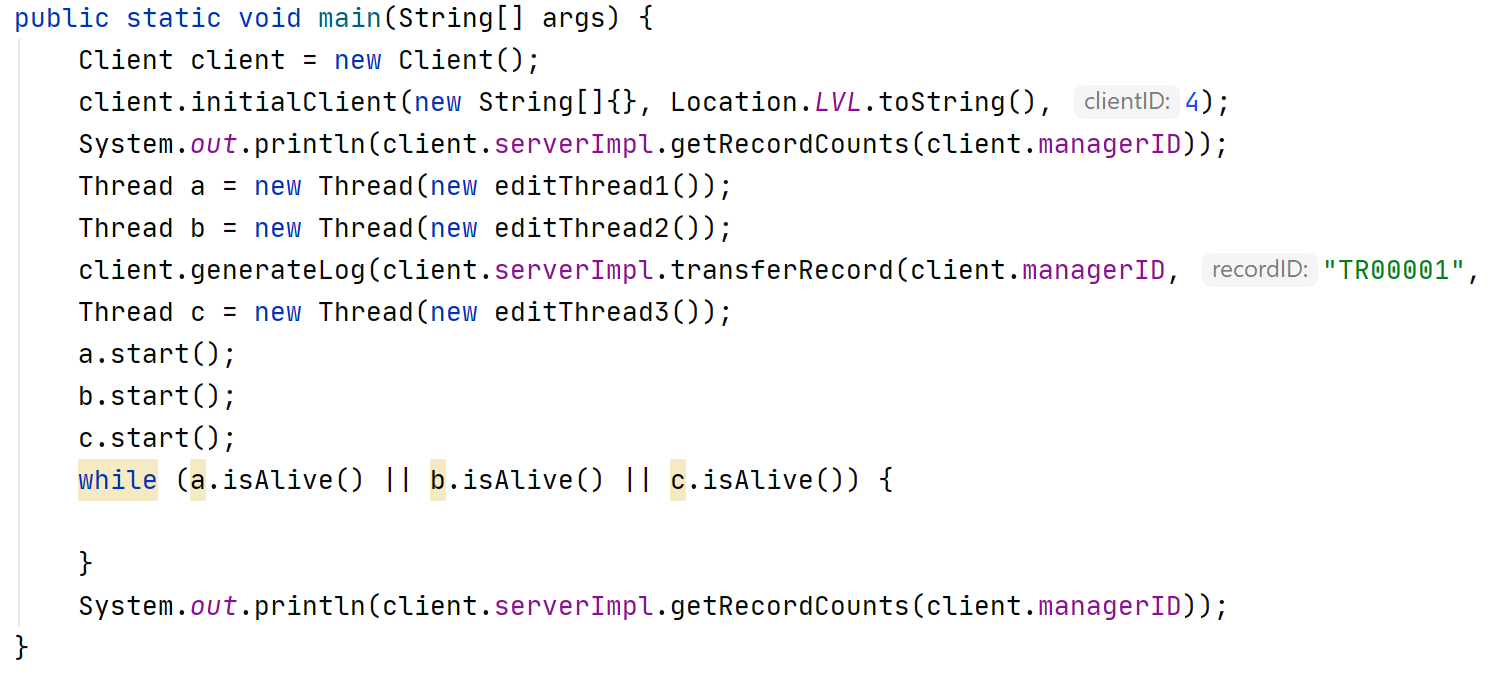
We can see the log message in LVL Server.



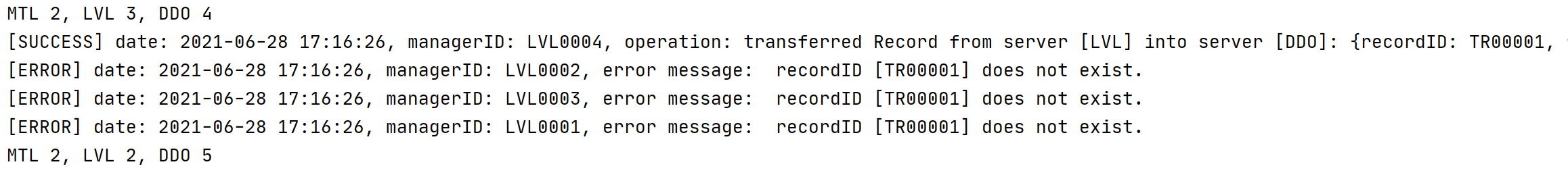
Contents in DDO Server.



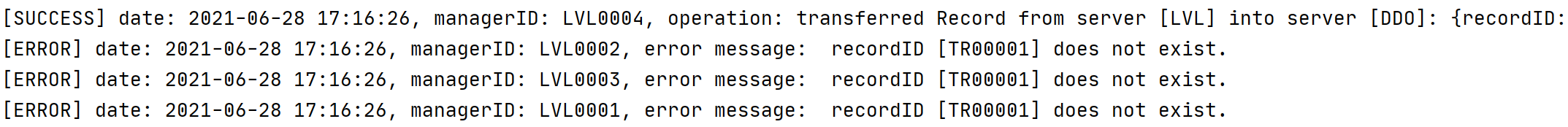
1. **This case tests the atomicity of the method in LVL Server. I start three threads to edit the field in the record, at the same time, I start a thread to transfer it. We can analysis the result whether is consistent through logs.**



We can see the log messages as below.



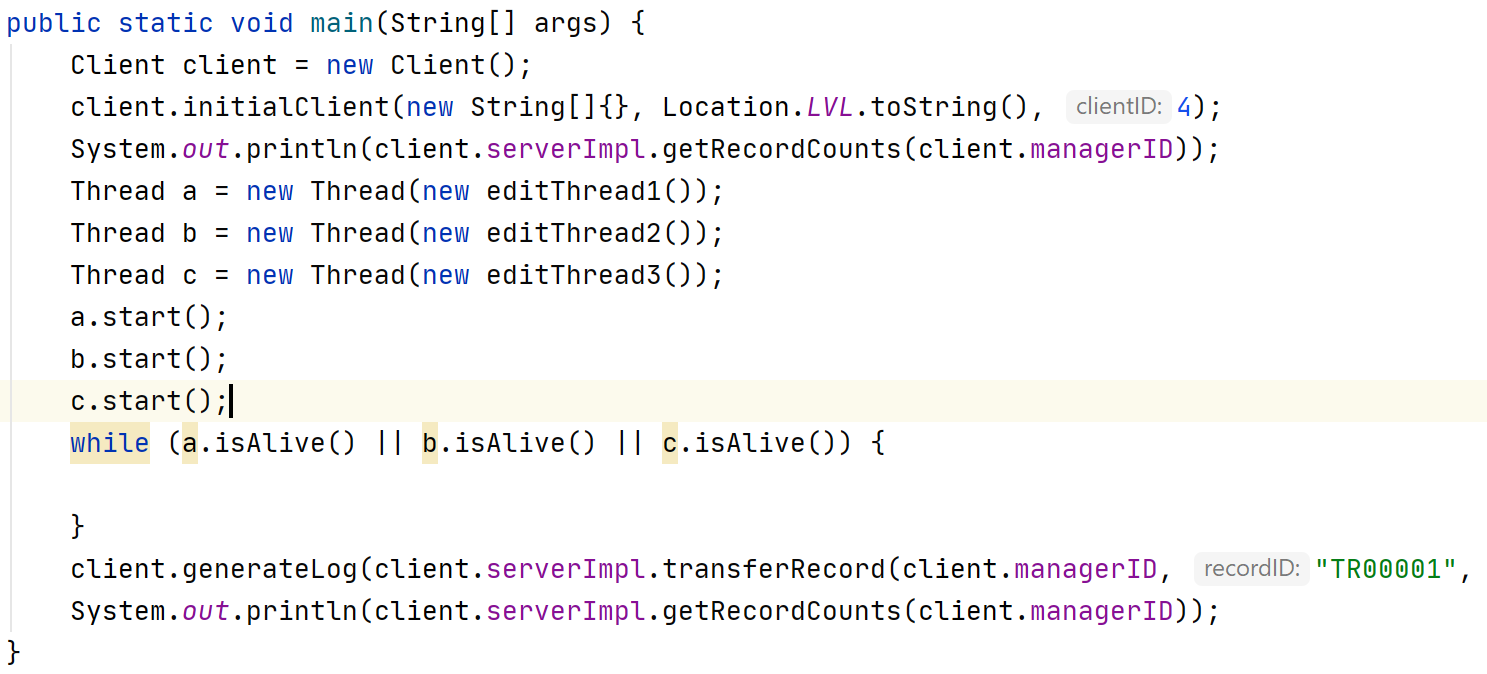
Log messages in LVL Server:



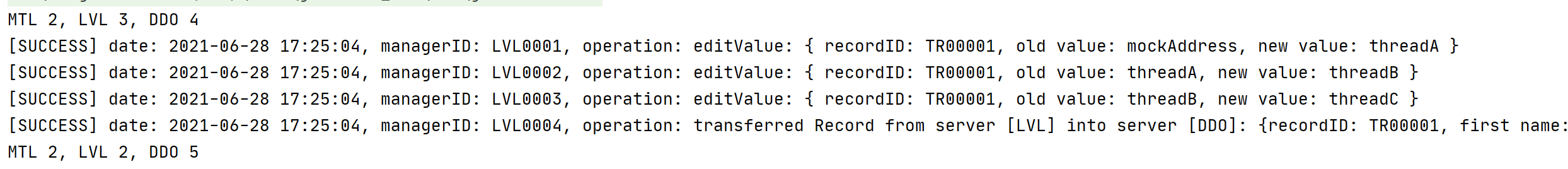
Log messages in DDO Server:



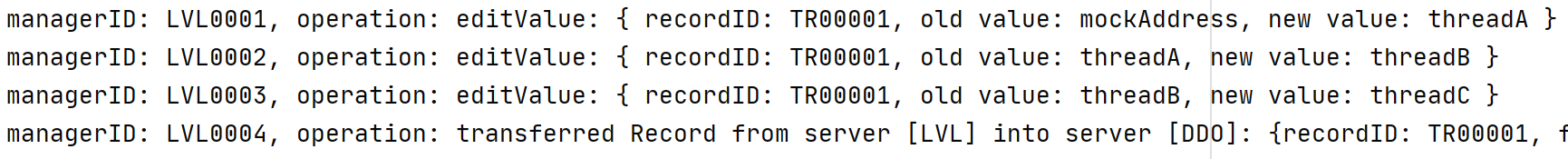
The content is consistent in the system. In the next step, we change the sequence of threads as below.



We can see the log messages as below.



Log messages in LVL Server:



Log messages in DDO Server:

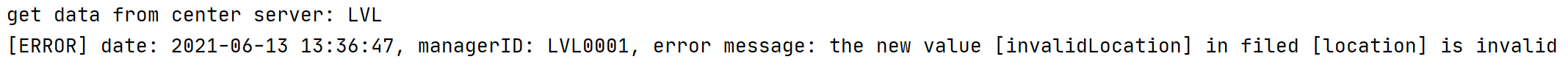


The content of the record is consistent in the whole system.

1. **We try invalid field value of “location” in LVL client.**



We will get data from LVL Server as below:



Log message in “LVL0001”:



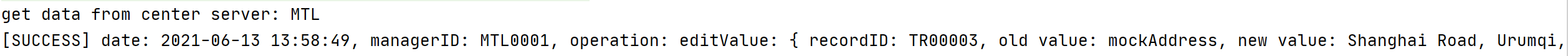
Same content can be seen in server log file “LVL”:



1. **We try new TeacherRecord’s value “address” in MTL client.**



We will get data from MTL Server as below:



New message is written in the log file “MTL0001” (the first half):



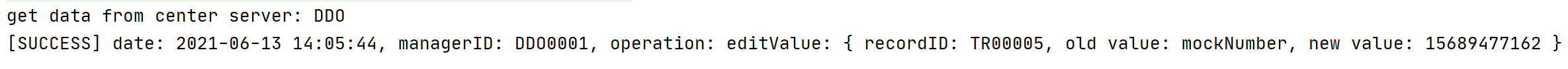
Same content in the server log file “MTL” (the second half):



1. **We try new TeacherRecord’s value “phone” in DDO client.**



We will get data from DDO Server as below:



New log file massage in “DDO0001” (first half):



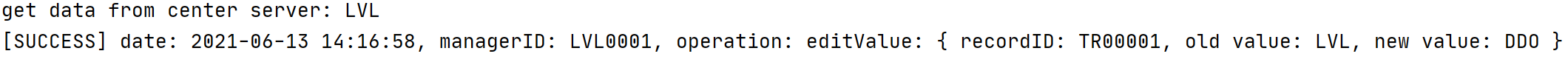
Same content in server log “DDO” (second half):



1. **We try new TeacherRecord’s value “location” in LVL client.**



We will get data from LVL Server as below:



Log message in “LVL0001” (first half):



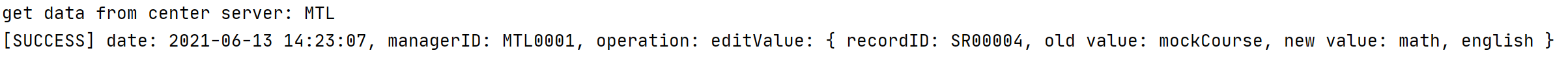
Same content can be seen in server log file “LVL” (second half):



1. **We try new StudentRecord’s value “courseRegistered” in MTL client.**



We will get data from MTL Server as below:



New message is written in the log file “MTL0001” (the first half):



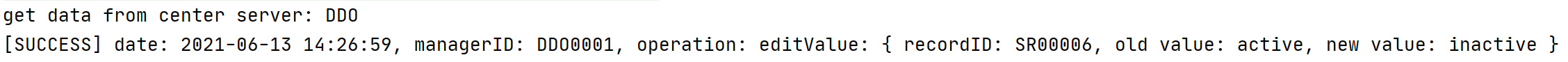
Same content in the server log file “MTL” (the second half):



1. **We try new StudentRecord’s value “status” in DDO client.**



We will get data from DDO Server as below:



New log file massage in “DDO0001” (first half):



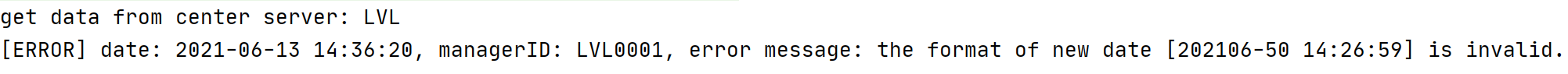
Same content in server log “DDO” (second half):



1. **We try new StudentRecord’s value “statusDate” in LVL client with invalid format.**



We will get data from LVL Server as below:



Log message in “LVL0001” (first half):



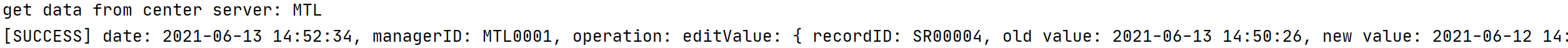
Same content can be seen in server log file “LVL” (second half):



1. **We try new StudentRecord’s value “statusDate” in MTL client.**



We will get data from MTL Server as below:



New message is written in the log file “MTL0001” (the first half):

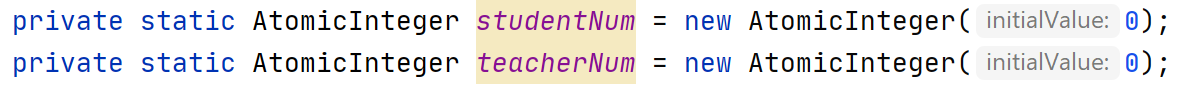


Same content in the server log file “MTL” (the second half):



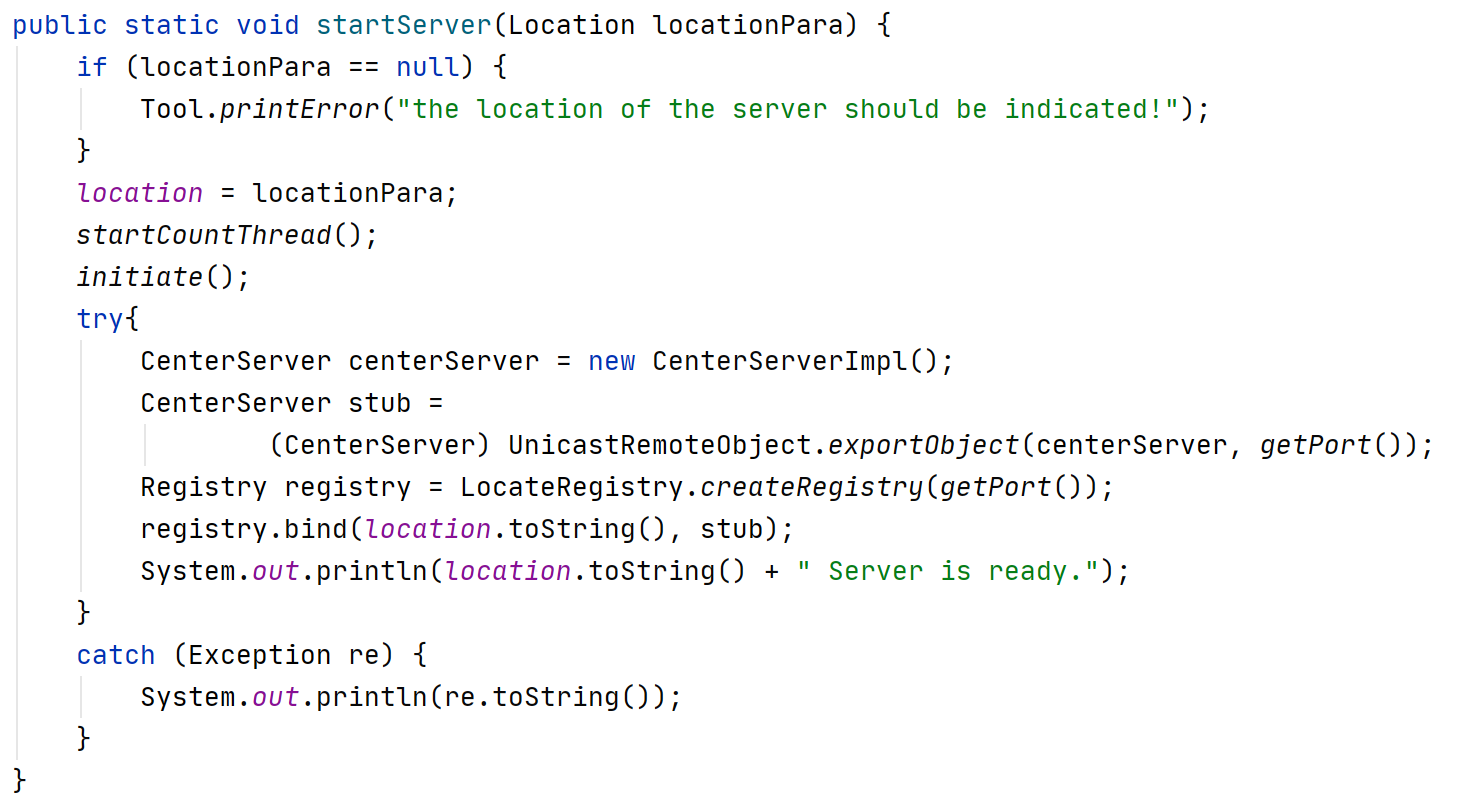
1. **Important parts:** 
   1. **IDServer**

I set up a new server to create unique recordIDs for new records. LVL, MTL and DDO Server have to communicate with IDServer to get the unique recordID when they want to create new StudentRecord or TeacherRecord. In IDServer, I use AtomicInteger class, which uses CompareAndSwap mechanism, to ensure the consistency of unique records. Compared with traditional synchronized methods, AtomicInteger has better performance. As we know, blocking and waking up threads are very time-consuming, and when some threads share the same AtomicInteger, they will operate its value in the memory directly and not be blocked. I use it to maximize the concurrency while ensuring the uniqueness of recordIDs.



**2. server startup method:**

We should indicate the location of the server at the beginning. Then the server will start Count Thread as mentioned above. At the next step, the server will set up some initial data and the context in the method initiate(). The server will register himself into the RMI server at last.



1. **Configuration:**

I store all configuration in the configuration file and load them into the Java Virtual Machine in the server startup. If we want to change the host address or port, we only need to change the content in the configuration file.



1. **Log function:**

We get the result and messages of the operation at first, then we add some extra information like date and managerID. At the last step, we print the message and write them into the log file whose directory and filename are stored in the configuration file.

