**COMP 6231 Project Report**

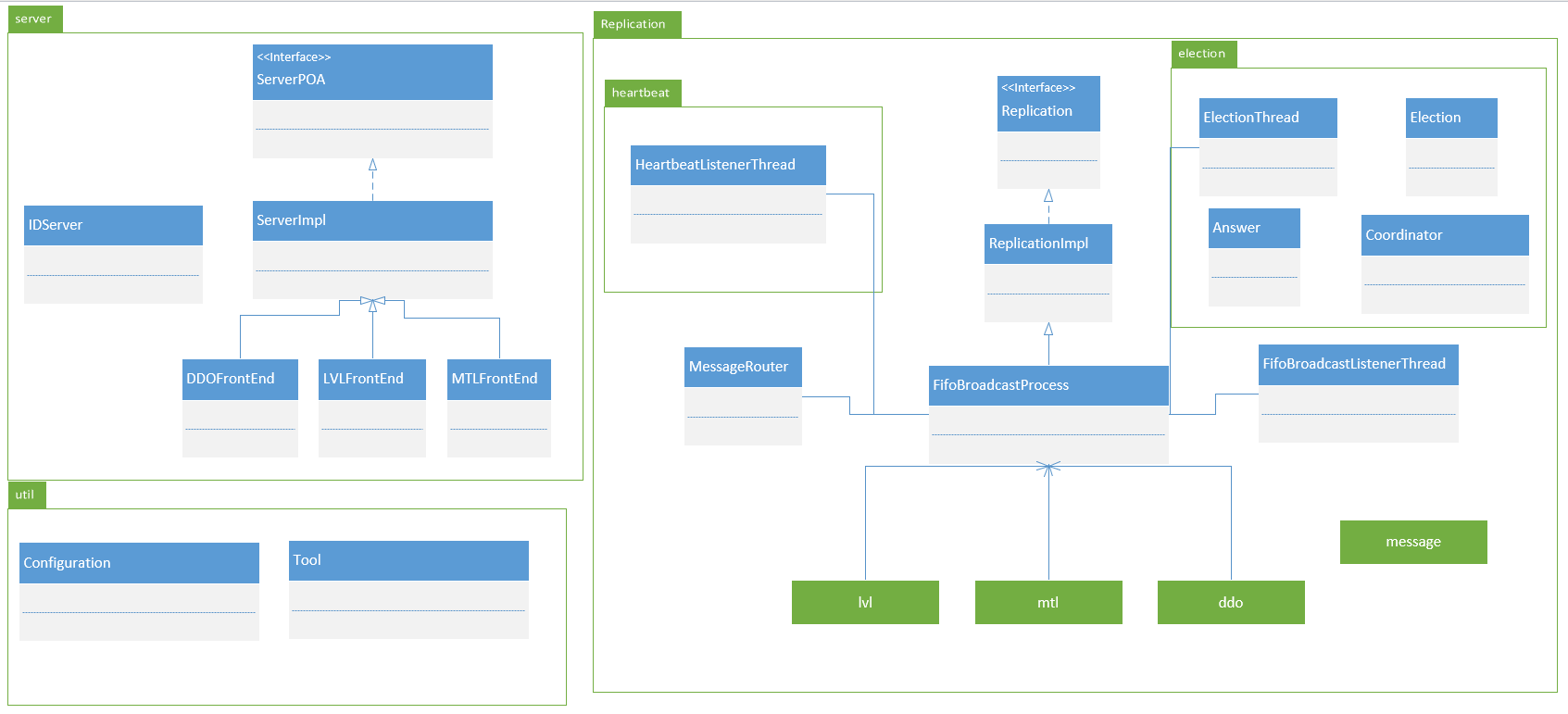
**Group Member:**

**Zexin Peng**

**Student number: 40166520**

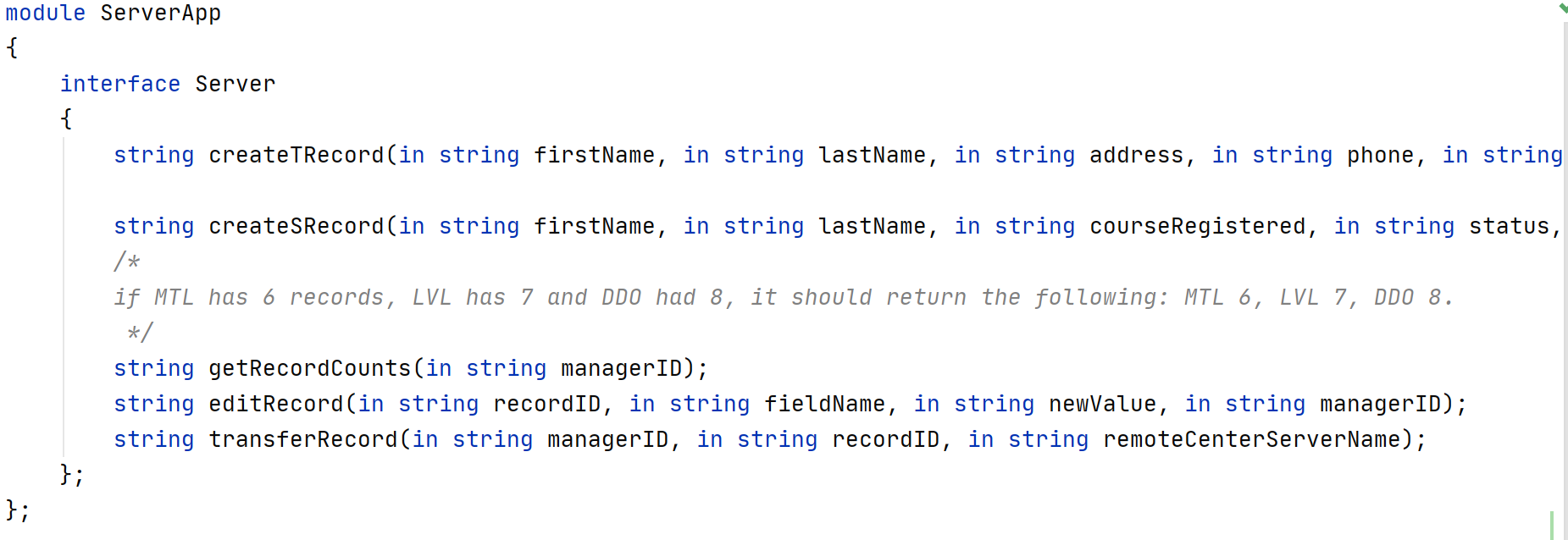
1. **Architecture Design:**

DDOFrontEnd, LVLFrontEnd and MTLFrontEnd are inherited from the same superclass “ServerImpl”, which implements the Interface “ServerPOA”. IDServer is responsible for assigning unique RecordID when creating new records. The responsibility of “Configuration” is to store some configuration information like ports and directory names. “Tool” provides basic support to other classes. The interface Replication in package replication defines some methods to deal with messages from the front end server, and class FifoBroadcastProcess and FifoBroadcastListenerThread can send, receive and process fifo broadcast messages together. Classes in package election implement the bully algorithm for leader election. HeartbeatListenerThread can deal with the heartbeat messages sent by other replications in the group periodically. Classes in package message are abstractions of different messages.

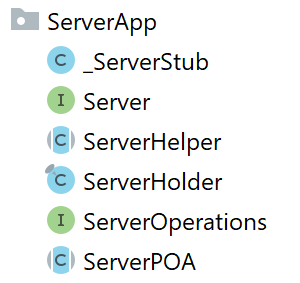


1. **Techniques：**
   1. **CORBA:**

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



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



* 1. **synchronized:**

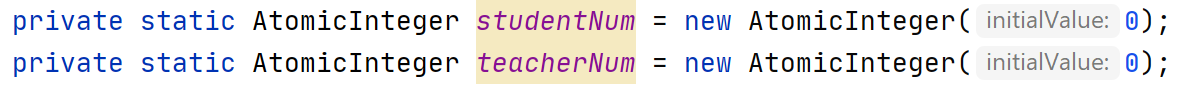
I use ‘synchronized’ in my code to avoid synchronization problem, because some shared data cannot be modified concurrently. By the way, to maximize concurrency, I do not maintain strong consistency in my assignment. In other word, if a server has 10 records, and a client will get 10 and will not be blocked 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. **Important parts:**

The implementation of configuration and log function is introduced in the last report, so I focus on the new part in this assginment.

* 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.



**3.2 failure detection:**

Every replication maintains a List to store the alive replications in the group, and sends heartbeat message using UDP to other alive replications in the list periodically, whose interval can be configured in Configuration file. I start a thread named HeartbeatListenerThread to listen the heartbeat messages from others and send a heartbeat message back. Furthermore, A replication will send heartbeat messages to others after it starts, and the replication which has received the message will add the newly started replication to its alive port list.



I use a ConcurrentHashMap to store the last received heartbeat message’s timestamp. Whenever a replication receives a heartbeat message, it will check whether there is any failure replication in the group. If the heartbeat message from an alive replication is timeout, it will remove the failed replication from the list.



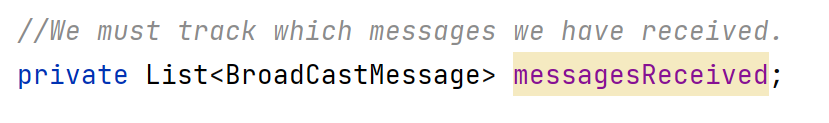
**3.3 leader election:**

There are three kinds of messages, election messages for starting election, answer messages for answering election messages, and coordinator messages for announcing header.

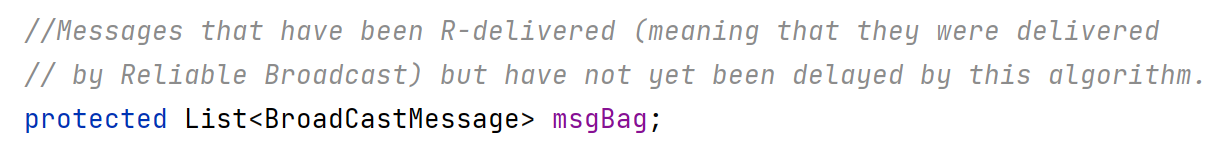
The replication will process first election after it starts. It will send election message to all replications with higher process ID if it does not have the highest process ID. If there is no answer message reply, it will be the leader and then send coordinator messages to other replications with smaller process ID.

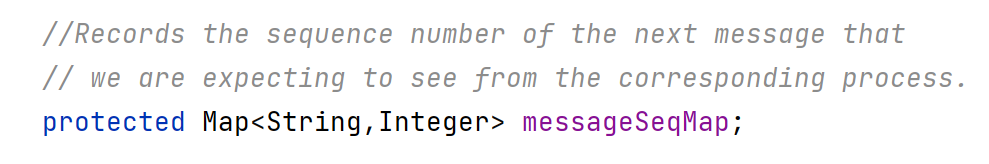
**3.4 fifo broadcast:**

The class BroadcastMessage is the abstraction of the messages sent by the fifoBroadcast systems. Each message has two value senderID and sequence number to uniquely define it. I start a new thread named FifoBroadcastListenerThread to receive broadcast messages. I store all accepted messages in a list, and only if the coming message is not in the list, then the listener can perform the following operations, otherwise, it will drop the coming message.



If the coming message is not in the messagesReceived list, the listener will send it to all replications in the group at first. At the next step, I put the message in a map, whose the key is the process ID and the value is sequence number of the message. Then I will check the sequence number to make sure the received message’s number is a bigger number than before. If not, it will be stored in a list.





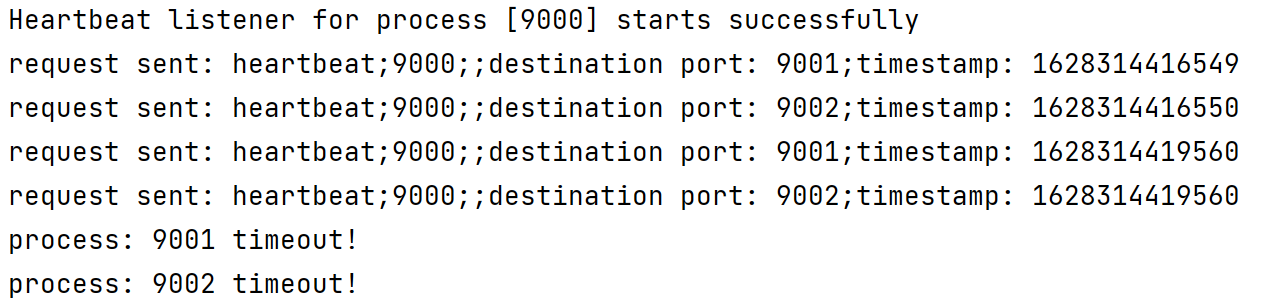
1. **Test scenarios**

I only demonstrate some simple cases in the report for simplicity, and I will show more complicated cases in the zoom presentation demo.

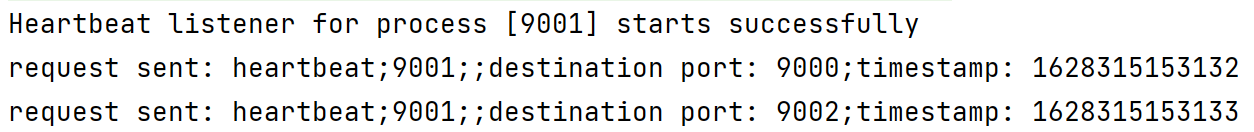
Each server initially has a few records except scenario 4.1. We have two records in MTL Server, three records in LVL Server and four records in DDO Server. The correctness of methods createTeacherRecord(), createStudentRecord(), getRecordCounts() and editRecord() are already tested in the report of assignment 1, so I focus on concurrency of my system in this report. All codes can be found in package client named TestScenario1, TestScenario2 and so on.

* 1. **Scenario 1:**

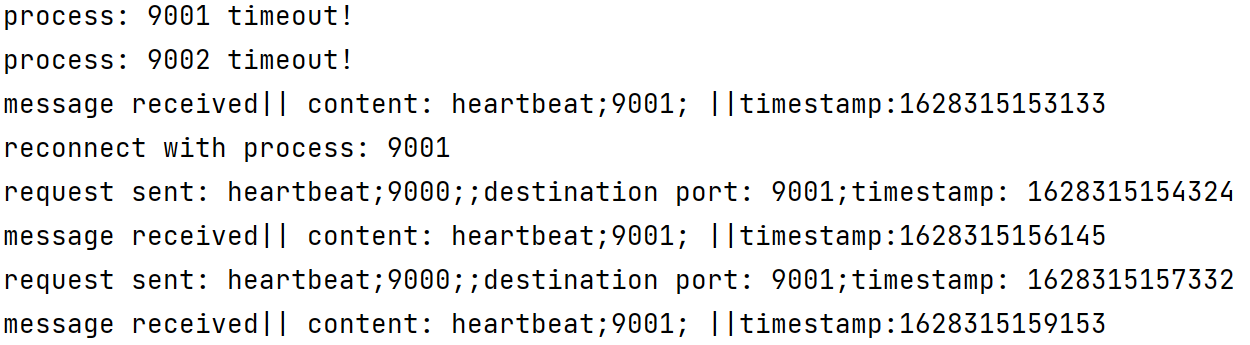
**Description:** This test will show the correctness of failure detection. I only start one process named 9000 at the beginning. Other replications ports in the group are 9001 and 9002. We can see that it sent four heartbeat messages to other replications and did not get any reply, so it removed them from the list.



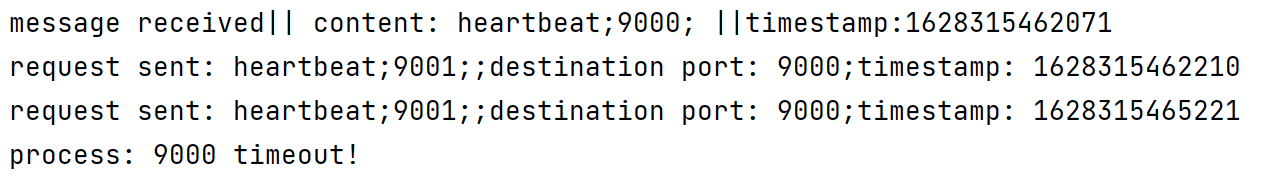
Then I start the replication named 9001, then it sent the heartbeat messages.



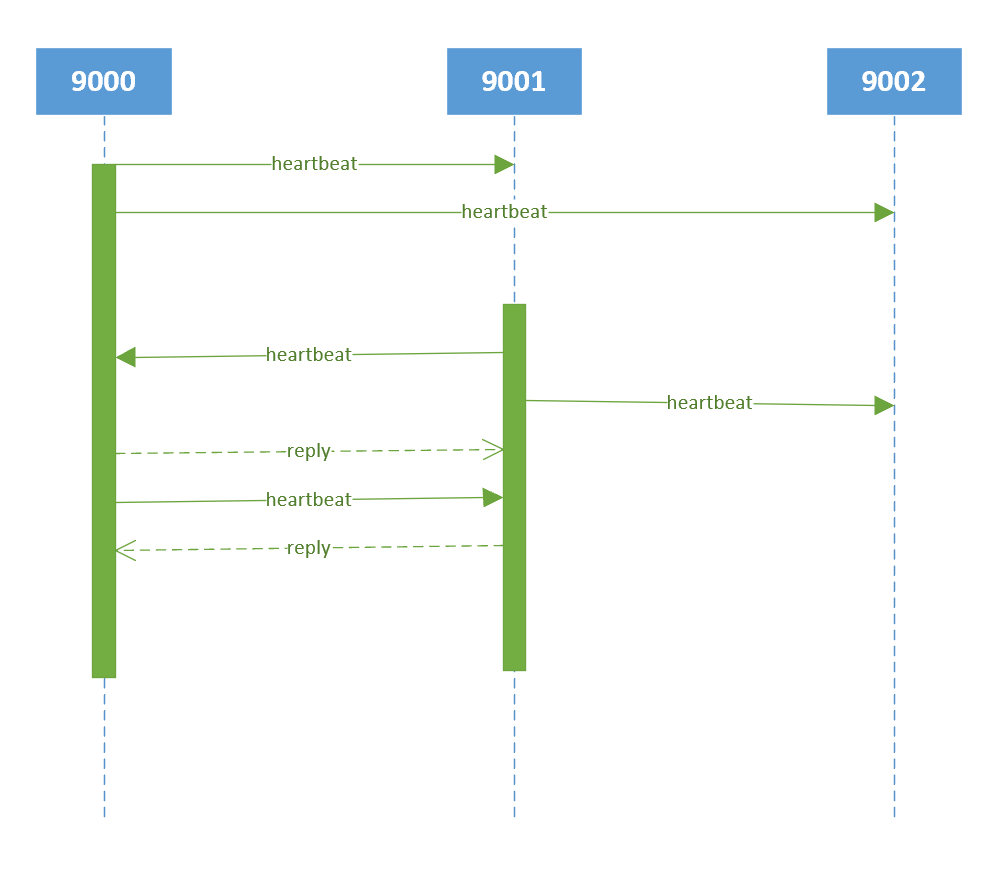
The replication 9000 received the message and maintain heartbeat messages connection.



Then I shut down the replication 9000, we can see logs in process 9001’s console.

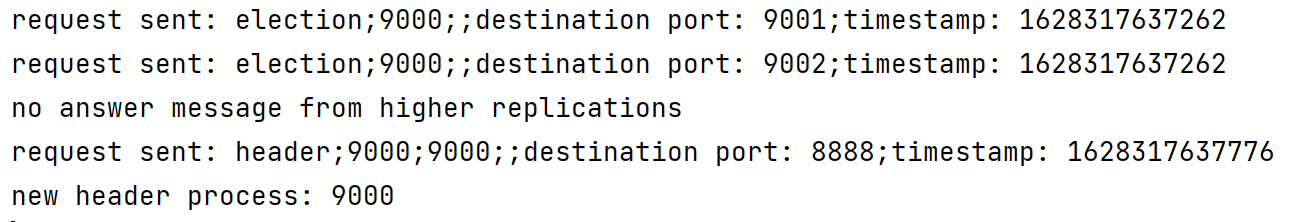


The dataflow shown as below.

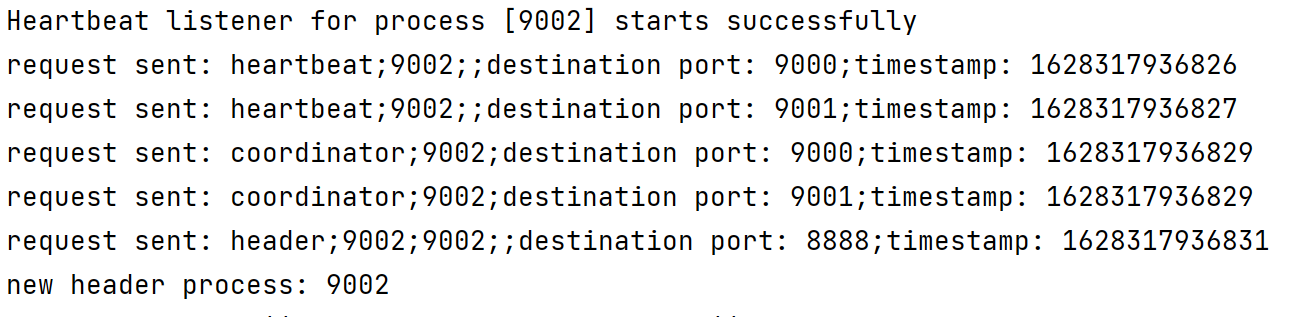


* 1. **Scenario 2:**

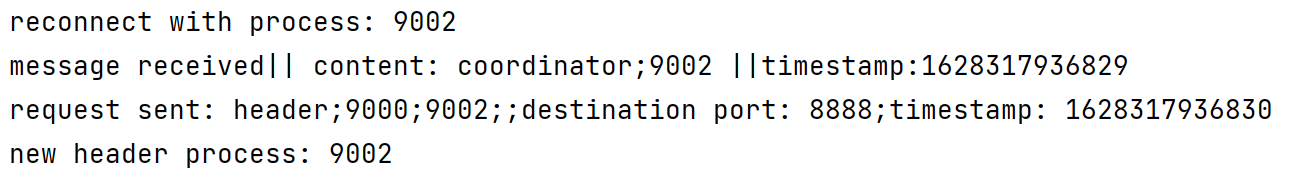
**Description:** This test will show the correctness of leader election. There are three kinds of messages, election messages for starting election, answer messages for answering election messages, and coordinator messages for announcing header. At the beginning, I started process 9000. It sent election messages to processes with higher processID, and it got no reply, so this process will be the leader.



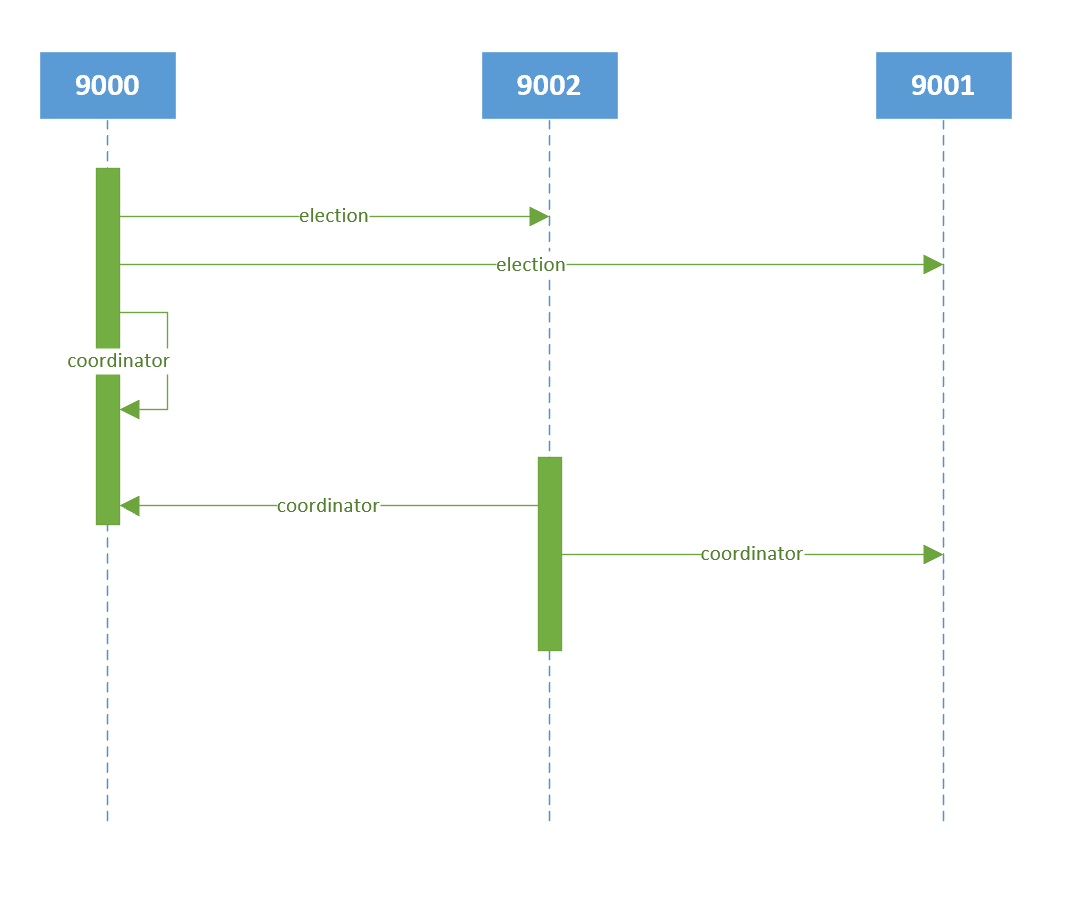
Then I start process 9001. It has highest process ID in the group, so it will be the leader.



At the same time, the process 9000 had received coordinator message, and set new header.



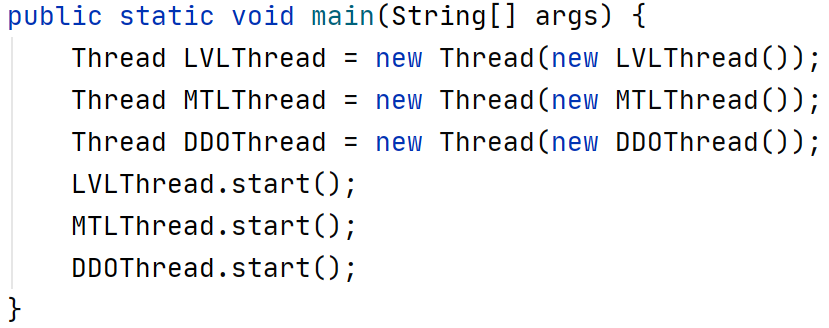
The dataflow shown as below.



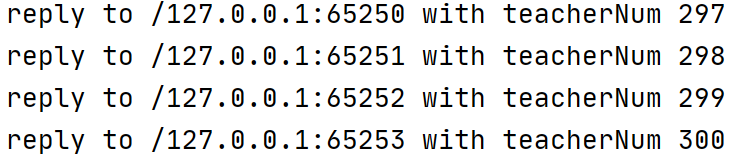
* 1. **Scenario 3:**

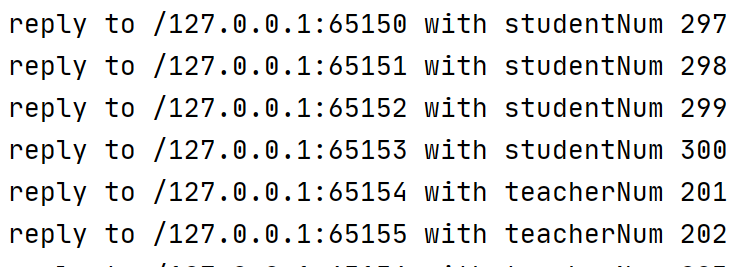
**Description:** I start three threads to create 300 student records and 300 teacher records concurrently to test the uniqueness of recordID and correctness of createTRecord and createSRecord. I also start three replications. You can see the code in client.TestScenario3.java.

Firstly, I create three threads and run them.

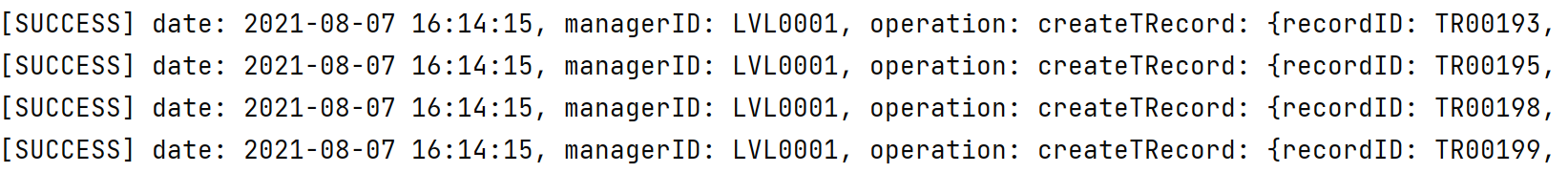


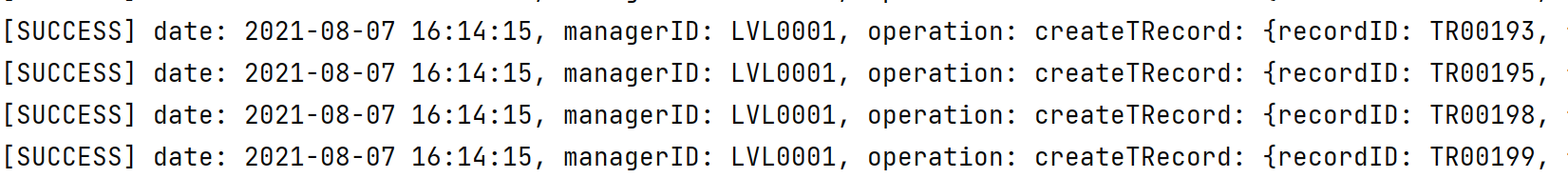
In the IDServer, I got logs below.



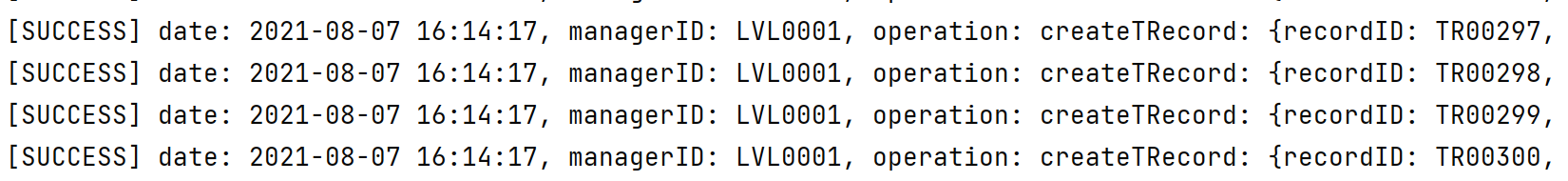


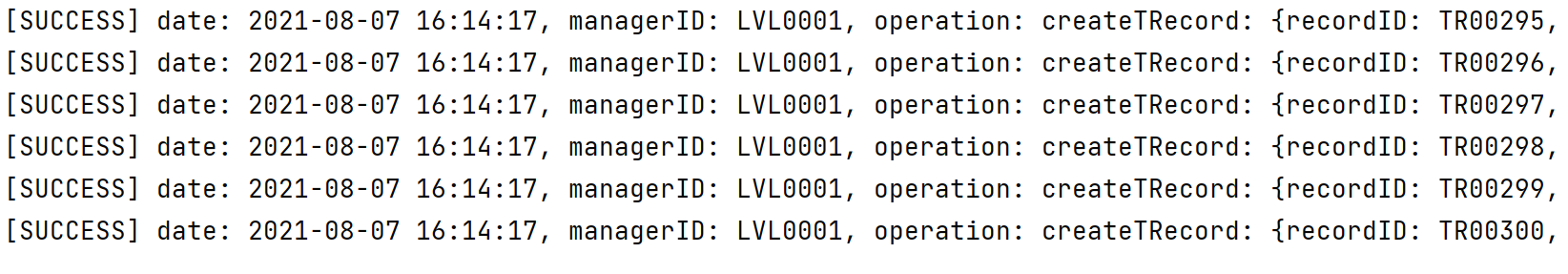
We can see logs in lvl replication1 and lvl replication2. They are in the same sequence.



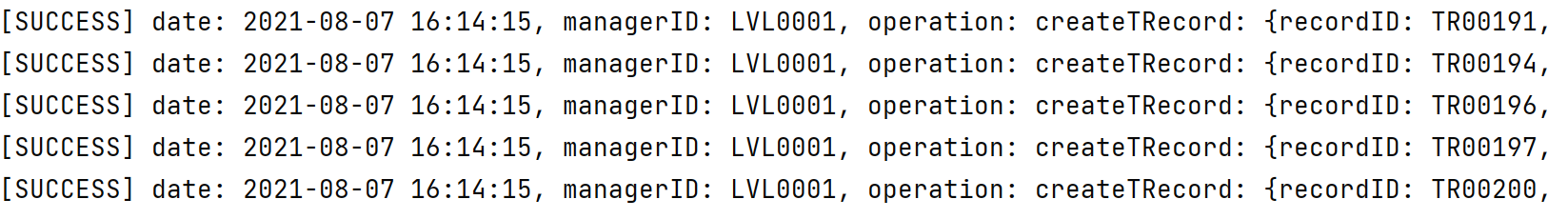


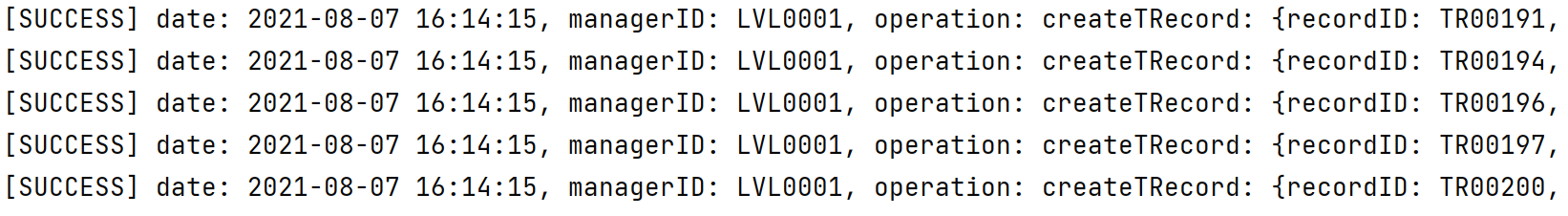
We can see logs in mtl replication1 and mtl replication2. They are in the same sequence.





We can see logs in ddo replication1 and ddo replication2. They are in the same sequence.

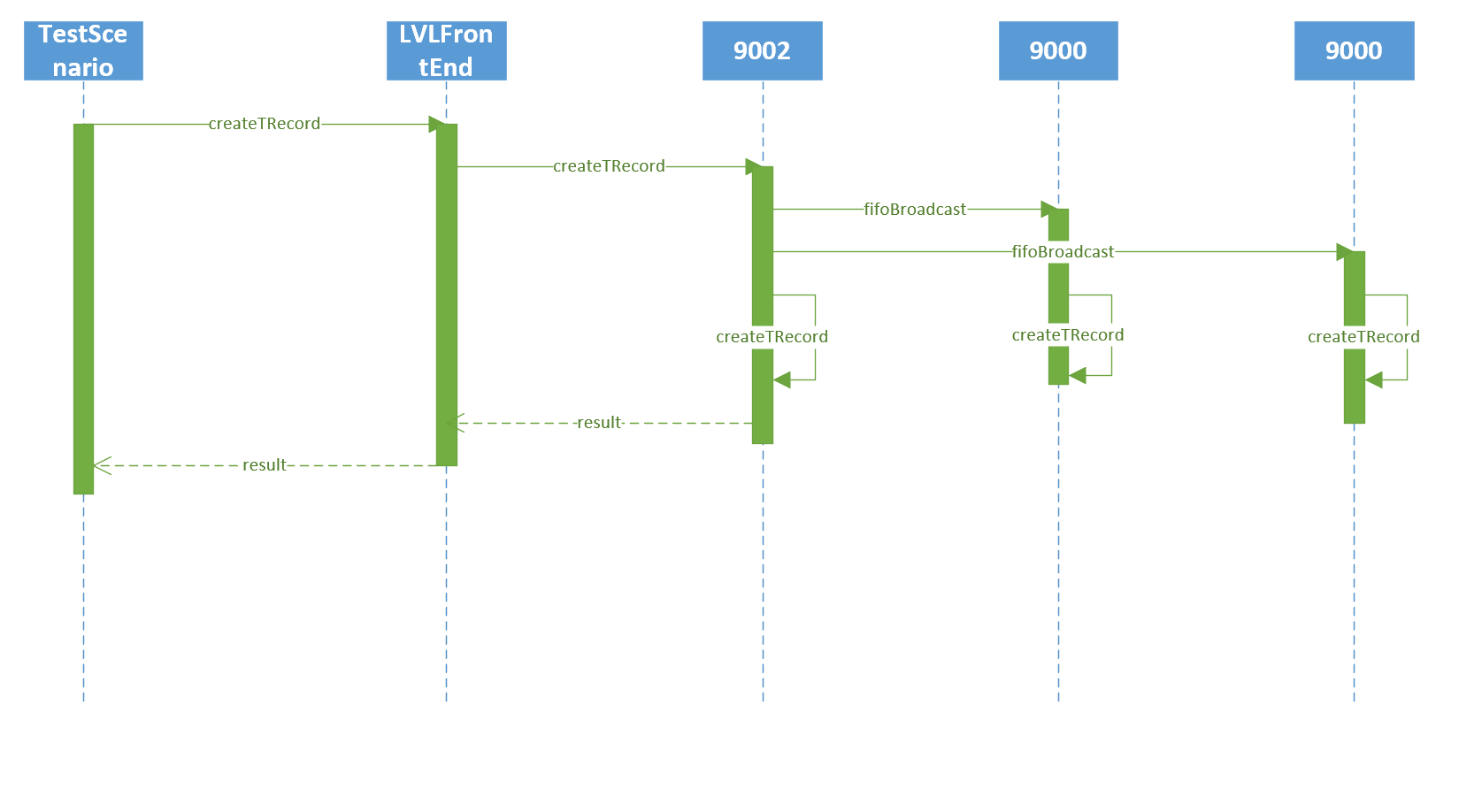




At last, I invoke the method getRecordCounts(), I got the result as below. The last student recordID is SR00300, and the last teacher recordID is TR00300. After analysis, we got a conclusion that all recordIDs are unique in the system, and the fifo broadcast system has done well.



The dataflow for createTRecord shown as below.



* 1. **Scenario 4:**

**Description:** this test scenario focus on editRecord() method in the server, and we can find it in client.TestScenario4a, 4b and 4c. At last, we will know the correctness of editRecord() method and log function.

1. **The recordID that does not exist.**



We will get log message in LVL replication1 and LVL replication2 as below





1. **I try to modify invalid field.**



We will get log message in LVL replication1 and LVL replication2 as below





1. **I try the valid operation.**

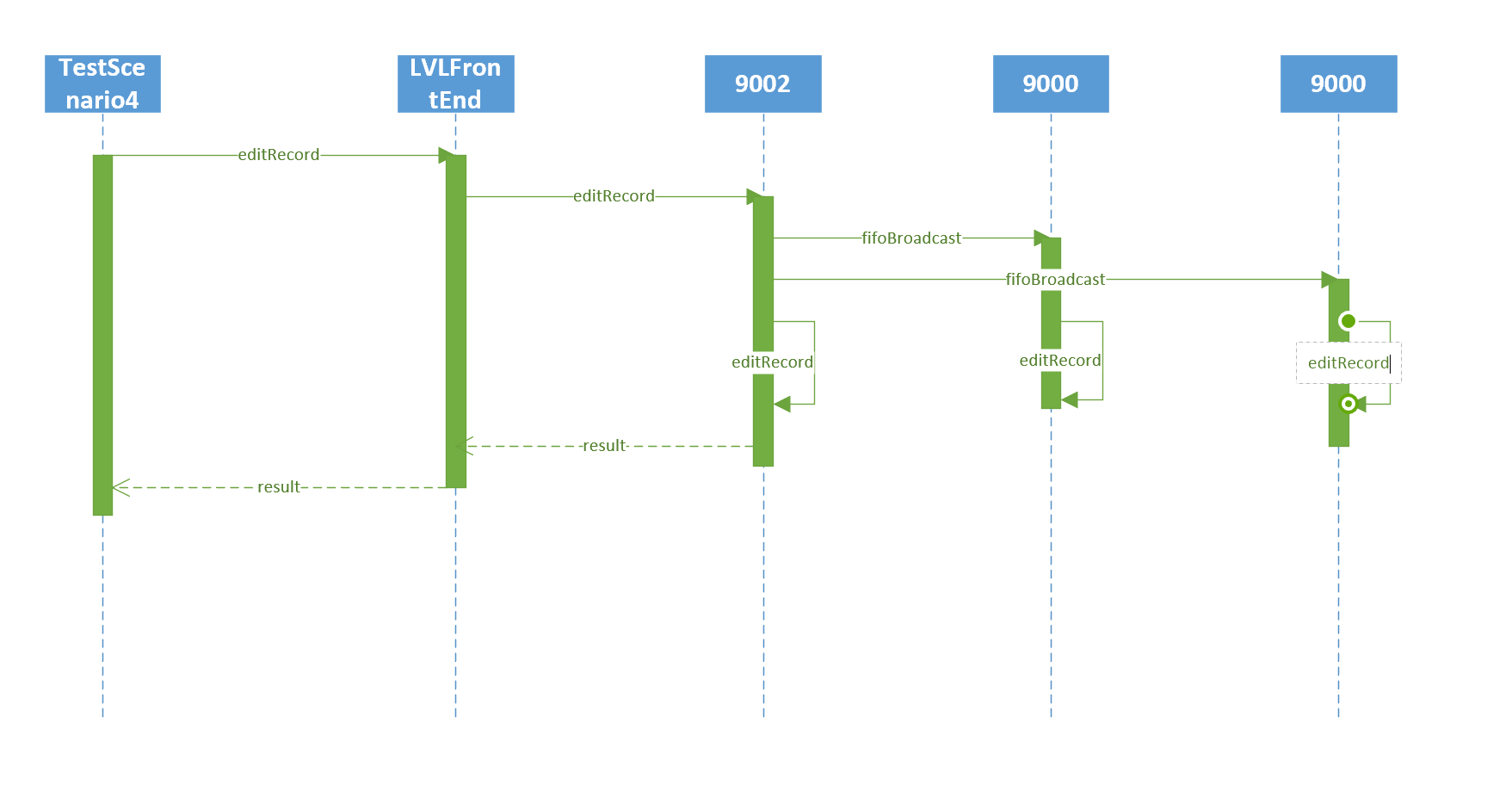


We will get log message in LVL replication1 and LVL replication2 as below





The dataflow shown as below.



* 1. **Scenario 5:**

**Description:** this test scenario focus on transferRecord() method in the server. By this test scenario, we will know the correctness of transferReocrd() method and log function. We can find it in client.TestScenario5a, 5b, 5c and 5d.

Firstly, we will try some invalid input.

**(a) The recordID that does not exist.**



We will get log message in LVL replication1 and LVL replication2 as below





**(b) I try the invalid location name.**



We will get log message in LVL replication1 and LVL replication2 as below



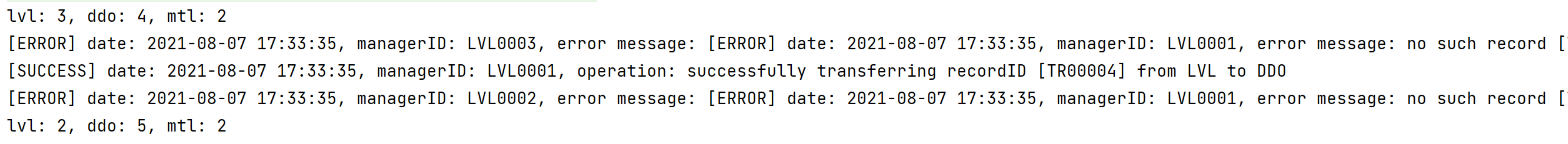


Secondly, we will test the correctness of the method, which focuses on atomicity and concurrency.

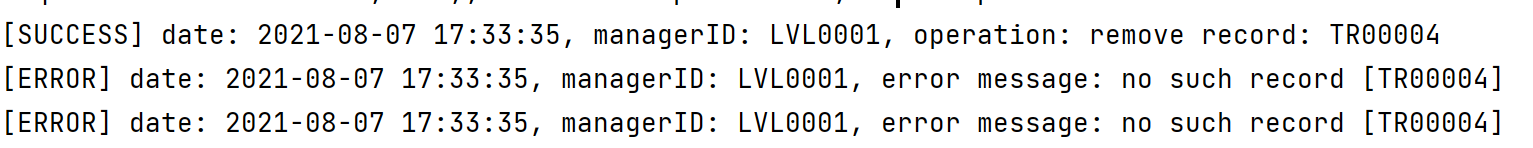
**(c) I test the atomicity of the method. I start three threads to transfer the record, and the record can only be transferred once.**



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



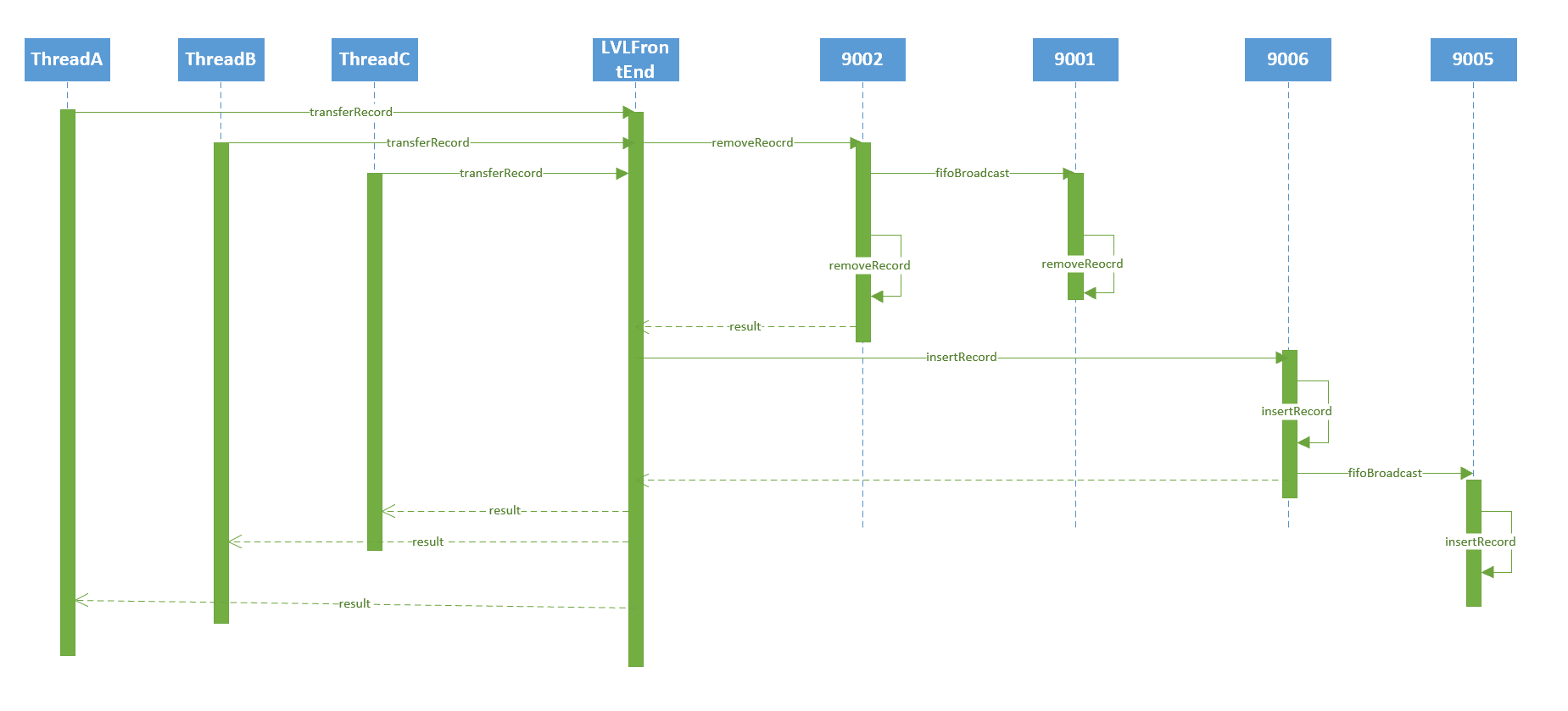
We got logs in LVL replication1 and LVL replication2



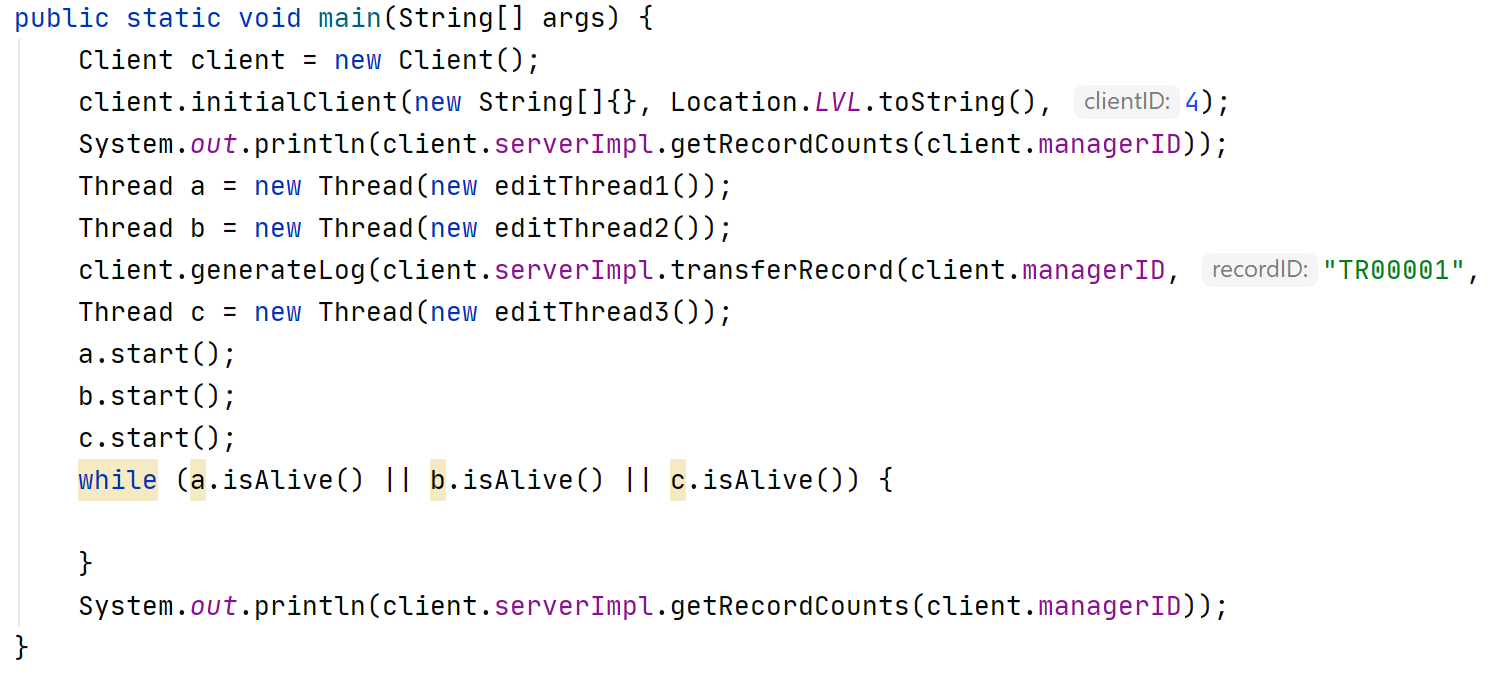
We got logs in DDO replication1 and DDO replication2



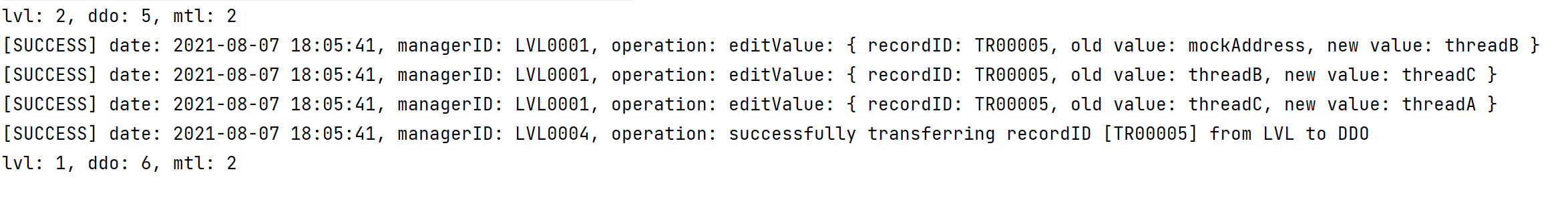
The dataflow is shown as below:



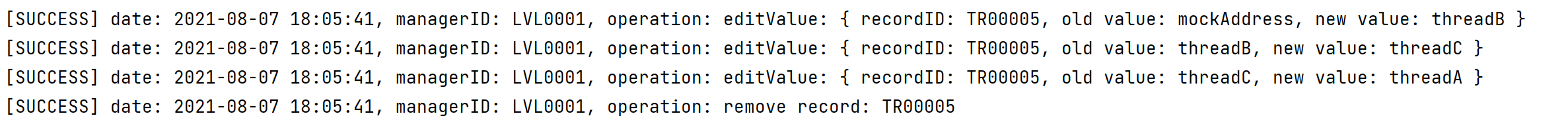
1. **This case tests the atomicity of the method. I start three threads to edit the field in the record, at the same time, I start a thread to transfer it. We can analysis the result whether is consistent through logs.**



We can see the log messages as below.



Log messages in LVL replication1 and LVL replication2



Log messages in DDO replication1 and DDO replication2:



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