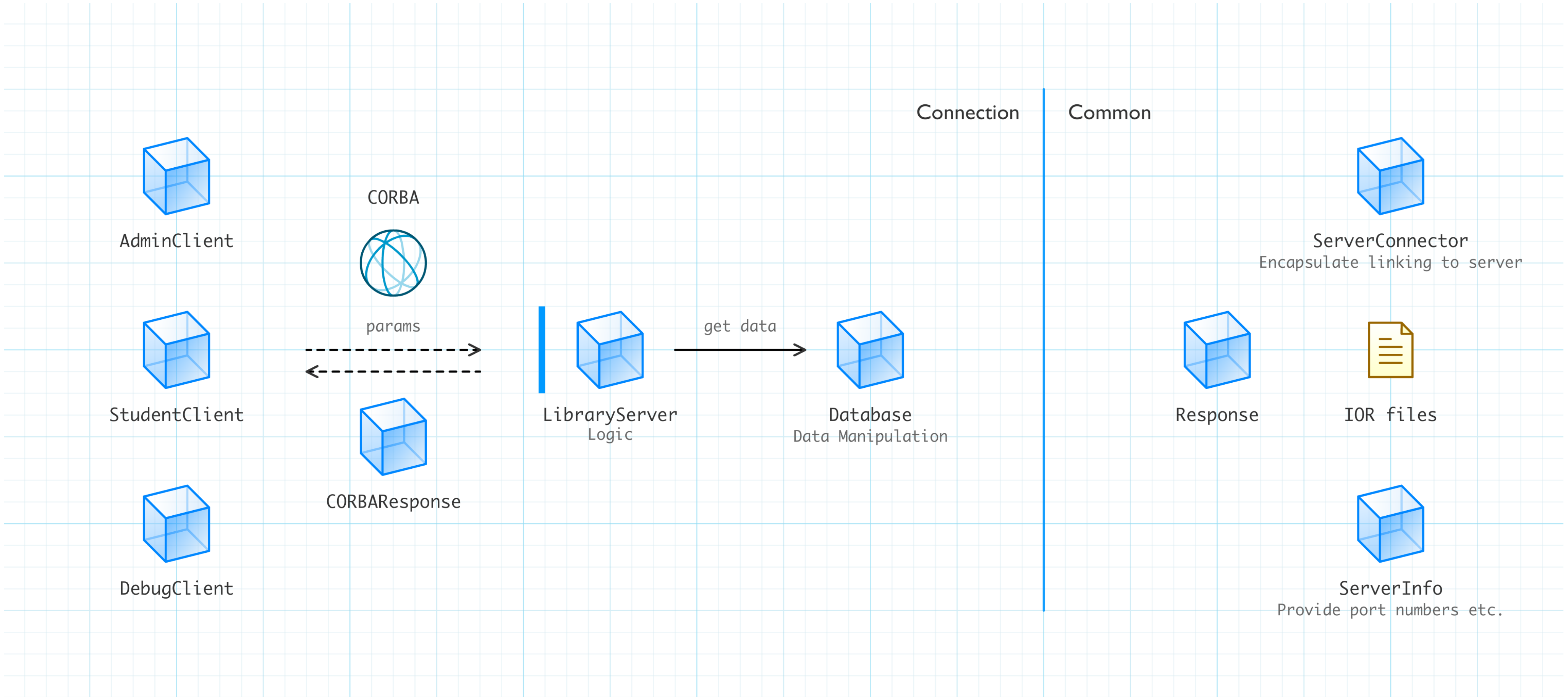
COMP 6231 Assignment 2 Documentation

Zhe Zhao

I. Importance Relationship between Classes / Object for Connection from Client to Server

This the a class/object relationship diagram, which is not a standard UML diagram. The details will be explained on the Page 2



Client & Server:

In this assignment, CORBA was use as the way to communicate between the client side and server side.

When `LibraryServer` is launched, it create an IOR file on the disk, so the clients will get the link to the server.

The interface was defined in the IDL file, which is not included in the graph.

Returning:

The return type of each methods defined is a `CORBAResponse` .

Because the `CORBAResponse` does not confirm to the Java Bean and it is not suggested to modify it or inherit it, `Response` was created.

Two methods `toCorba` and `setFromCorba` was created to build a bridge between `Response` and `CORBAResponse` to give the ability to transfer object between them easily.

When server execute, an `Response` was created. It will be transfer to a `CORBAResponse` object before returning.

When client get the response, it get the `CORBAResponse` . But, it will be transferred to a `Response` object.

Both `Response` and `CORBAResponse` contain the same attributes `errorCode` and `data` , but the `Response` provide other methods like `isSuccess` .

Client:

`StudentClient` & `AdminClient` are to different executable console based application for user to use the system.

`DebugClient` , as a new client in this assignment, is a new console based application for debugging and checking information about the server.

`ServerConnecter` is a class to encapsulate the way of getting stub object form server. In this assignment. It get a CORBA stub and provide the object back to the clients to connect.

Server:

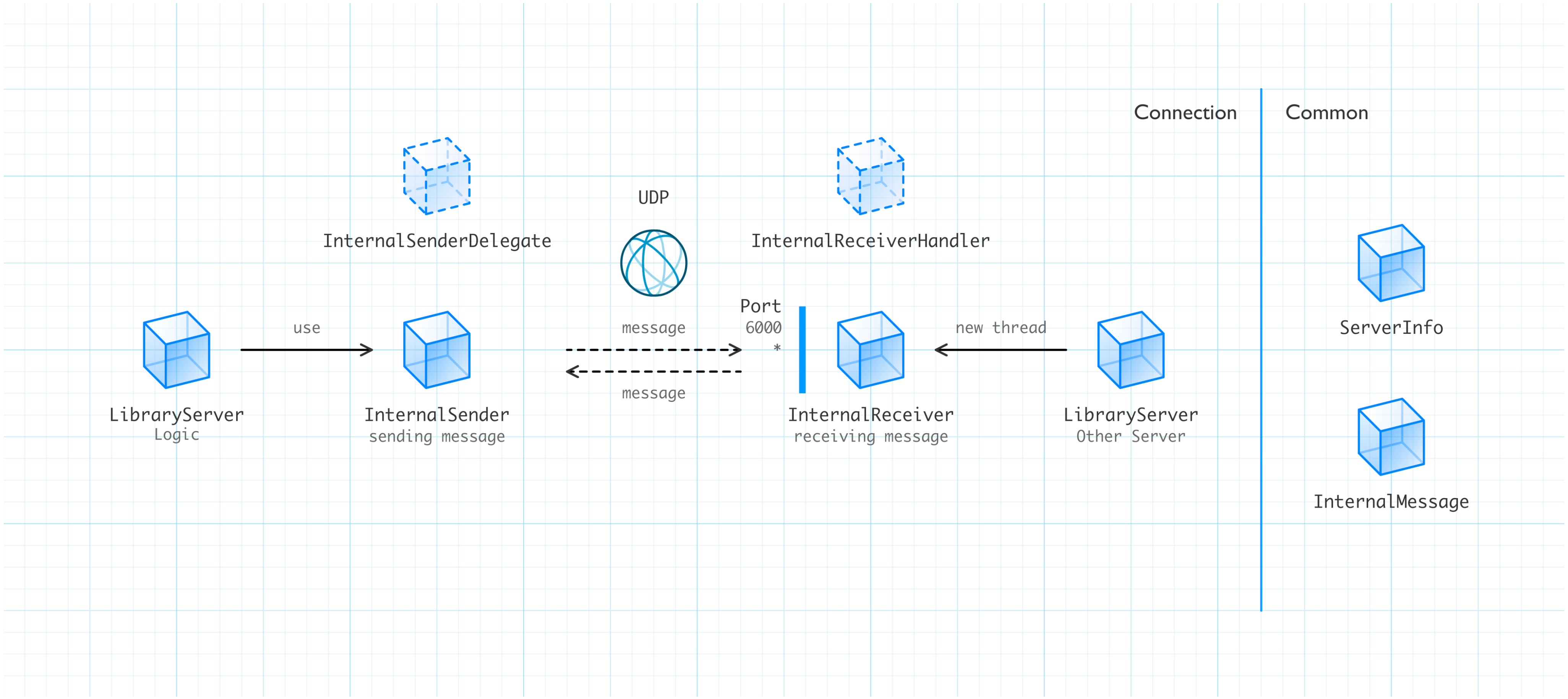
`LibraryServer` inherit the `LibraryServerPOA` .

It manage the request from the clients and give a response back.

`Database` will behave like a data base but an object, providing methods to manipulating data.

2. Importance Relationship between Classes / Object for Connection between Servers

This the a class/object relationship diagram, which is not a standard UML diagram. The details will be explained on the Page 4



Server to Server

In this assignment, UDP was used as the way to communicate between different servers.

Each `LibraryServer` has two object `InternalSender` and `InternalReceiver` to handle the communication between servers.

Although the message sent are string type via the UDP, `InternalMessage` was created to handling the marshalling and unmarshalling job.

Internal Message:

`InternalMessage` is a class to describe the message through the servers. A server will send an `InternalMessage` to other server, and the other server will give back an `InternalMessage` as return.

It has two attributes:
`type` : Identify the request type, so the other server will know what to do. Of the type marked as a return.
`parameters` : tells the value names and values in the message.
So therefore, the return message will may also have multi-params.

Actually, `InternalMessage` will not be send directly via the UDP, but a string comes from an `InternalMessage` object.

`InternalMessage` has two methods to handle the marshalling and unmarshalling:
`encode` : marshall the information to string.
`decode` : unmarshall a string back to the message object.

Internal Sender:

`InternalSender` , as an attribute from the `LibraryServer` , take the duty of sending messages to other server and get feedbacks.

When `InternalSender` received a returning message from an other server, it will call a `InternalSenderDelegate` object, which will be the `LibraryServer` , to handle how to deal with the feedbacks.

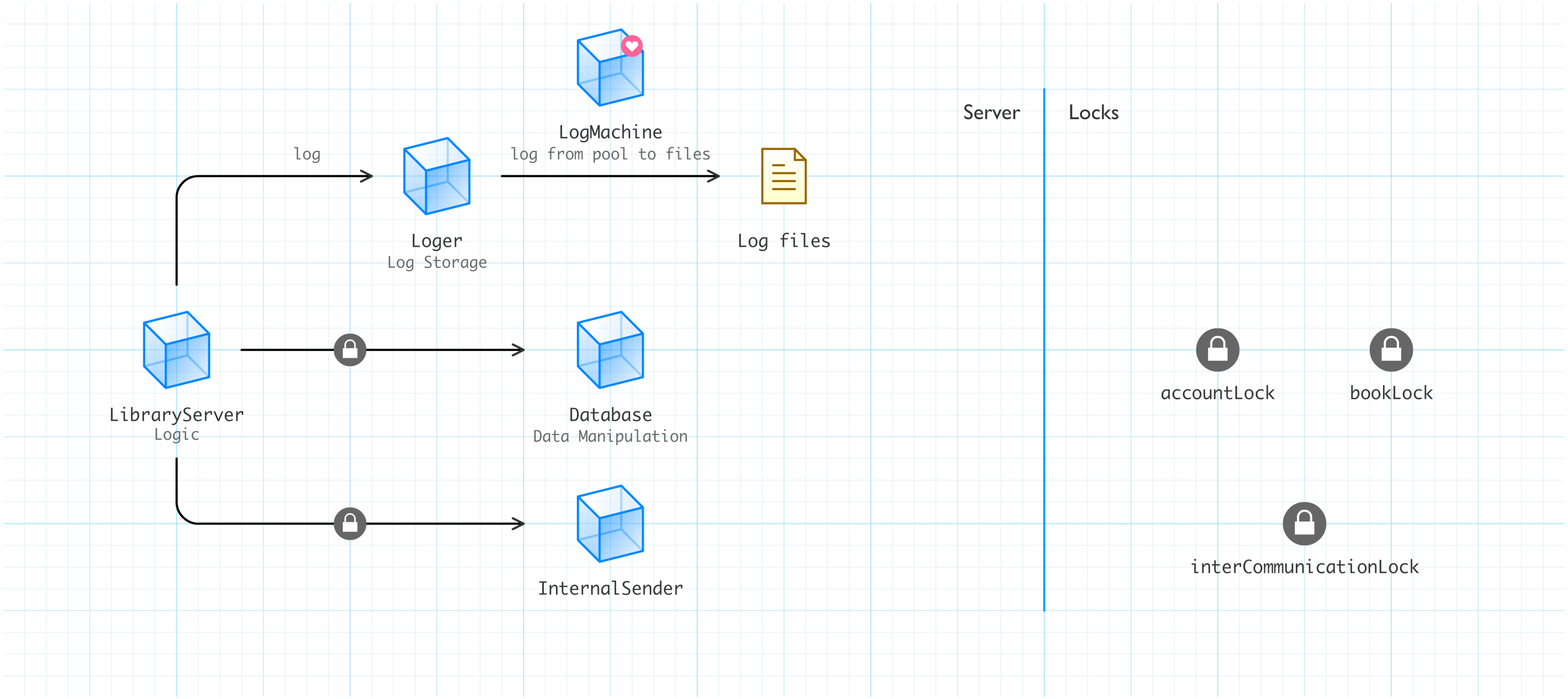
Internal Receiver

When `LibraryServer` was launched, an `InternalReceiver` will be created as a thread to wait messages come from other server.

`InternalReceiver` store some `InternalReceiverHandler` objects. Each `InternalReceiverHandler` will be associated with a message type. When a message comes in, `InternalReceiver` will call a `InternalReceiverHandler` object to handle the message.

3. Importance Relationship between Classes / Object for Server

This the a class/object relationship diagram, which is not a standard UML diagram. The details will be explained on the Page 6



Synchronization & Lock:

The synchronization work will be handled in the `LibraryServer` object not the `Database` object.

Every methods are following the steps below:

1. Validate the parameters. Check the permission.
2. Synchronize by using a specific lock.
3. Manipulate the data.
4. Log the record.
5. Unlock the code.

Lock:

There are different lock objects inside the `LibraryServer` object, to maximize the concurrency of the code:

1. A lock for Books
2. A lock for Accounts
3. A lock for inter communications.

In this way, a client trying to rent a book will not affect some other client who is creating an account, because they are using different locks.

Log:

Because IO is a time consuming work, every time when `LibraryServer` record a log, it will be sent into a log pool.

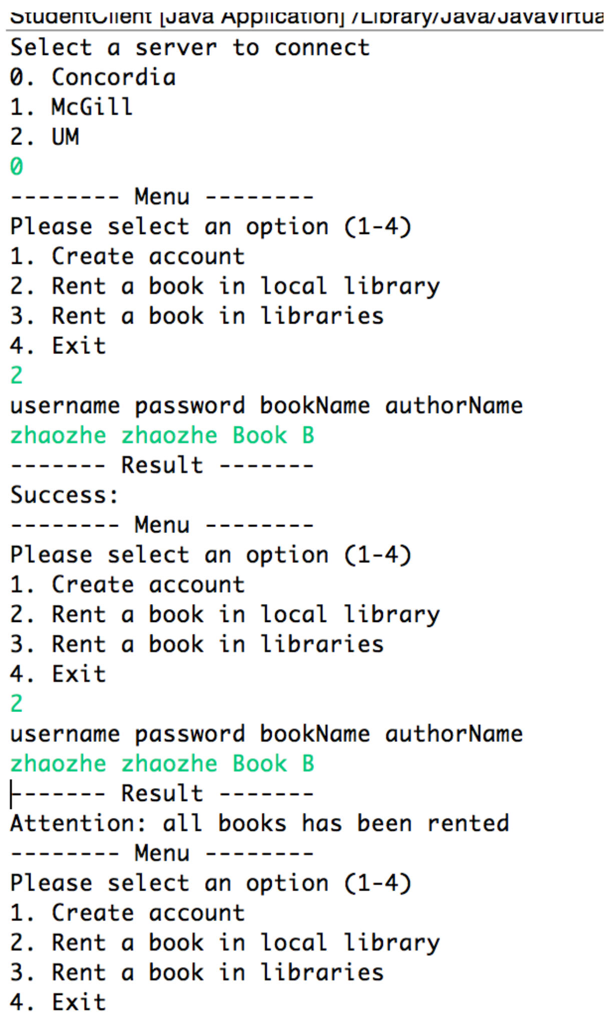
`LogMachine` will check the pool every certain minutes to write the log information on to the disk.

So, this reduce the time in some `@synchronized` code.

4. Test

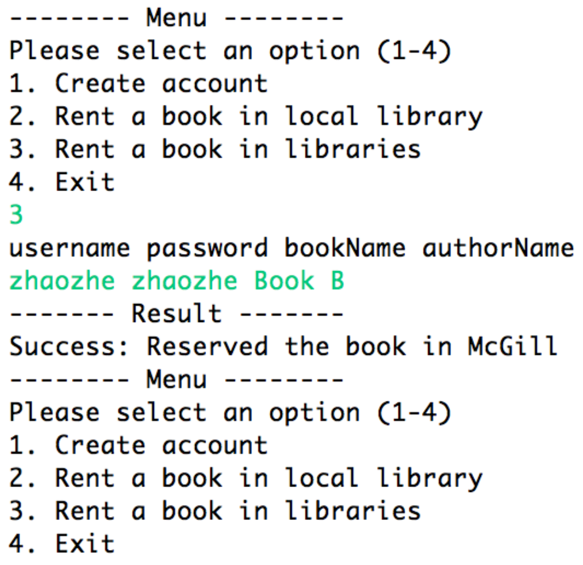
Each server has one book “Book” and an account “zhaozhe”

When the user first rent the book, it success, the second time failed. because there is no book in the local library



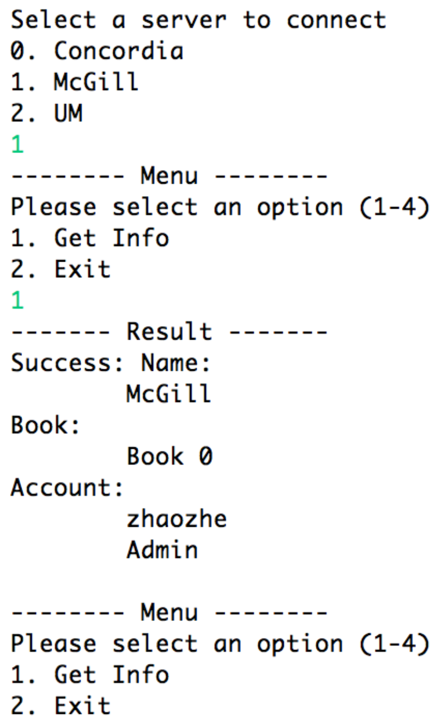
```
StudentClient [Java Application] /Library/Java/JavaVirtua
Select a server to connect
0. Concordia
1. McGill
2. UM
0
-------- Menu --------
Please select an option (1-4)
1. Create account
2. Rent a book in local library
3. Rent a book in libraries
4. Exit
2
username password bookName authorName
zhaozhe zhaozhe Book B
------- Result -------
Success:
-------- Menu --------
Please select an option (1-4)
1. Create account
2. Rent a book in local library
3. Rent a book in libraries
4. Exit
2
username password bookName authorName
zhaozhe zhaozhe Book B
------- Result -------
Attention: all books has been rented
-------- Menu --------
Please select an option (1-4)
1. Create account
2. Rent a book in local library
3. Rent a book in libraries
4. Exit
```

Then tried to rent from other library, success.



```
-------- Menu --------
Please select an option (1-4)
1. Create account
2. Rent a book in local library
3. Rent a book in libraries
4. Exit
3
username password bookName authorName
zhaozhe zhaozhe Book B
------- Result -------
Success: Reserved the book in McGill
-------- Menu --------
Please select an option (1-4)
1. Create account
2. Rent a book in local library
3. Rent a book in libraries
4. Exit
```

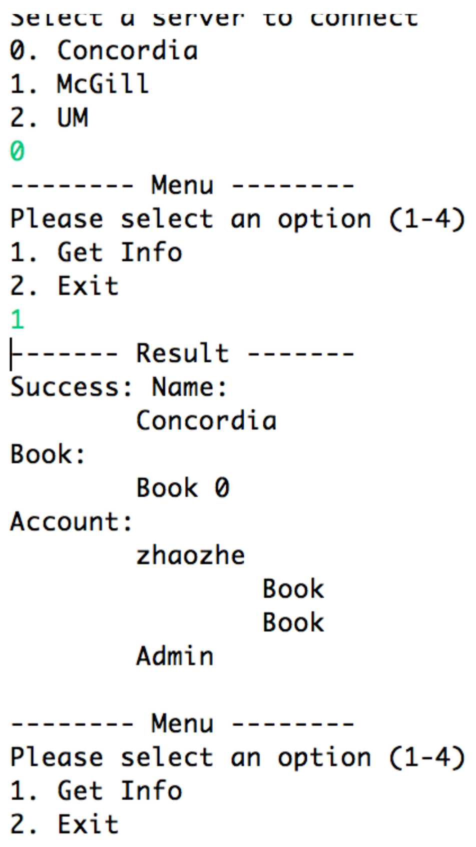
Then check the book in Mcgill, no more book.



```
Select a server to connect
0. Concordia
1. McGill
2. UM
1
-------- Menu --------
Please select an option (1-4)
1. Get Info
2. Exit
1
------- Result -------
Success: Name:
        McGill
Book:
        Book 0
Account:
        zhaozhe
        Admin

-------- Menu --------
Please select an option (1-4)
1. Get Info
2. Exit
```

Then check the accounts in Concordia, A book locally rented, and remotely rented was recorded.



```
Select a server to connect
0. Concordia
1. McGill
2. UM
0
-------- Menu --------
Please select an option (1-4)
1. Get Info
2. Exit
1
------- Result -------
Success: Name:
        Concordia
Book:
        Book 0
Account:
        zhaozhe
                Book
                Book
        Admin

-------- Menu --------
Please select an option (1-4)
1. Get Info
2. Exit
```