COMP 6231 – Assignment 2

Distributed Library Management System using Java IDL (CORBA)

By Yogesh Kumar

Winter, 2019

Table of Contents

Introduction

This is a modified/updated version of DLMS in assignment 1. Just like that one, we need to design and develop a distributed library management system consist of different libraries distributed on a local network. A distributed system is a system which consists of multiple components distributed over the network but work together as a single unit of software.

Just like Assignment1, we have three libraries in this assignment as well, namely, Concordia Library, McGill Library and Montreal Library. Each one be replica of each other in sense of functionality but have their own individual databases and information regarding books and library users and other things. We have all the functionality which part of Assignment 1 but also added another functionality to library servers where user can exchange the book which he currently posses with another book which he wants to borrow following all the requirement/conditions of assignment 1.

We will develop this system using Java IDL which is java implementation of CORBA architecture. With the help of Java IDL (CORBA), an object running in one Java Virtual Machine can invoke methods on another object running in another Java Virtual Machine, just like Java RMI but the major difference is that in Java RMI, both the applications have to be implemented in Java but in CORBA architecture we can have heterogeneous implementation of communicating ends. We are running different components such as three library instances, user client, manager client and central repository on different Java Virtual Machines and with the help of Java IDL technology, they will be communicating with each other. So even being distributed into multiple components, the whole system act as a single unit and work as one system. Again for clients, we have used JavaFx technology to build the GUI for user/manager interaction.

As shown in *Figure1.*, the whole application can be divided into components which are described briefly below:

<u>CORBA (Central Repository Interface and Server Interface)</u>

This component is specific to Java CORBA implementation and is the core to this application. It has interface definition language, IDL, files and CORBA related file which are generated using those IDLs. Also the functionality defined by the java interfaces generated by compiling IDL files are implemented by different classes of library servers to provide method definition to those functionalities.

<u>Central Repository</u>

This component act as a repository for other components. When this component starts, It fetches the Naming service from ORB and register itself with a defined name so that others can call its methods remotely using that stub name. It holds the information for the library servers regarding like stub details and also the UDP socket connection details with which different library servers communicate with each other. Whenever a library server starts, it fetches the stub of central repository from ORB naming service and registers its details with the central repository so that the clients can get the required details from this repository to connect and communicate with required library server.

Clients

These components are almost similar as in Assignment 1 apart from the fact that now they are working using CORBA and are developed using JavaFx. These are GUI components which let users/managers interact with the different libraries and perform different operations.

User client is used by library users to perform operations such as borrow books, return books and find items where as Manager client is used by the library managers to add new items, delete items or list available items. These clients get library servers details from the central repository stub and then further using ORB naming service fetches library stub and invoke methods on those stubs so that appropriate action can take place.

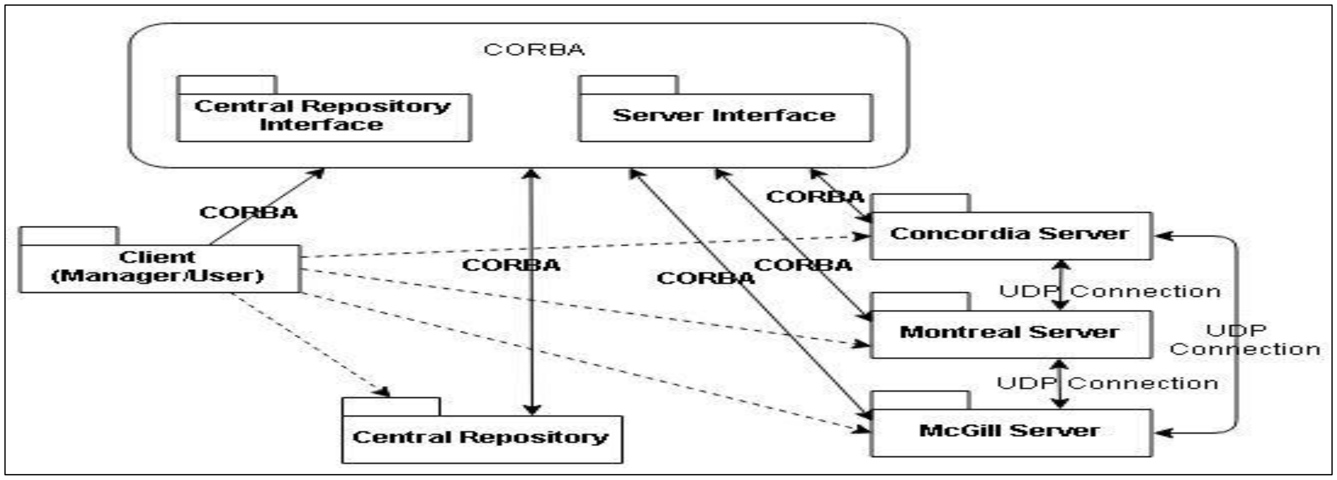


Figure 1. Over all component view of whole system.

Concordia Library/ McGill Library/ Montreal Library

They are similar in functionality as in Assignment 1 but now uses CORBA just like everything else and also have some added functionality. These components represent different libraries, running independently from each other. They have their own database holding information regarding registered users/ managers, books in library, waiting list for books, borrowed items and so on. They also communicate with each other using UDP connection when a user from one library wants to perform some operation on another library.

Whenever a server starts, its details, like port number on which stub is registered and stub name and UDP socket connection details etc., are registered with the central repository for the clients to use that information and communicate with the libraries. And also for different libraries to get the UDP socket details from central repository to communicate with other libraries over UDP socket connection.

There is two type of communication here. One is at application layer, through CORBA technology, which happen between clients, central repository and libraries. And second type of communication is UDP socket connection based communication, which is at transport layer, between libraries.

System Design

Architecture

• CORBA Architecture:

CORBA architecture basically shows how CORBA(ORB) act as the mediator between client and server components and help them to communicate. Due to this indirection and CORBA's own IDL we get remote method invocation ability between heterogeneous applications.

As shown in *Figure2,* there are java files which are generated by compiling IDL files using idlj compiler. These java files include mediator which are used to communicated between client and remote server components and interfaces which are implemented by remote components.

All the remote components (Central repository, Concordia Server, McGill server and Montreal Server) register their stubs with the ORB naming service and implement required interfaces generated by IDL files. Client application interact with ORB and fetch the stubs using mediator files and services provided by CORBA and execute the methods on server components remotely.

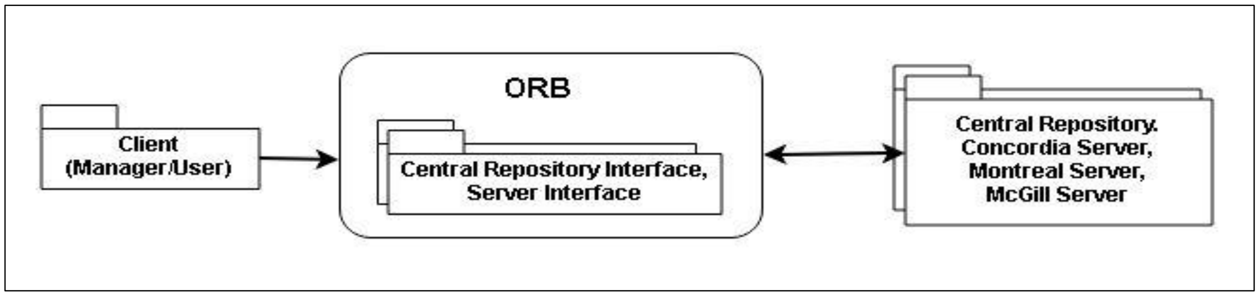

ORB
Client (Manager/User)
Central Repository Interface, Server Interface
Central Repository, Concordia Server, Montreal Server, McGill Server

Figure 2. CORBA Architecture Diagram

• Application Architecture:

The application architecture is same as in Assignment 1 but using CORBA for implementation instead of Java RMI and is explained below:

The architecture of this distributed application is MVP(model-view-presenter) which is a which is a variation of MVC(model-view-controller) as, unlike MVC, in this architecture model is not responsible for updating the view which is JavaFx based rather controller is responsible to do so once it receives the result from the remote methods which are exposed by objects running on library servers, it invokes through CORBA stubs.

As shown in *Figure3.,* we have two clients, manager client and user client, which have JavaFx based view which is backed by respective controllers. Users/managers can interact with whole distributed application using these clients and GUI based view. Through these views, event is triggered by users which are captured by the controller classes. Based on the type of event triggered and parameters passed by GUI user , appropriate server details are fetched from the central repository which for these sort of requests act as model. Once controller have details of library server's, it invokes the methods on the library server model classes. These model class then make local calls to the in memory database class to do the computation.

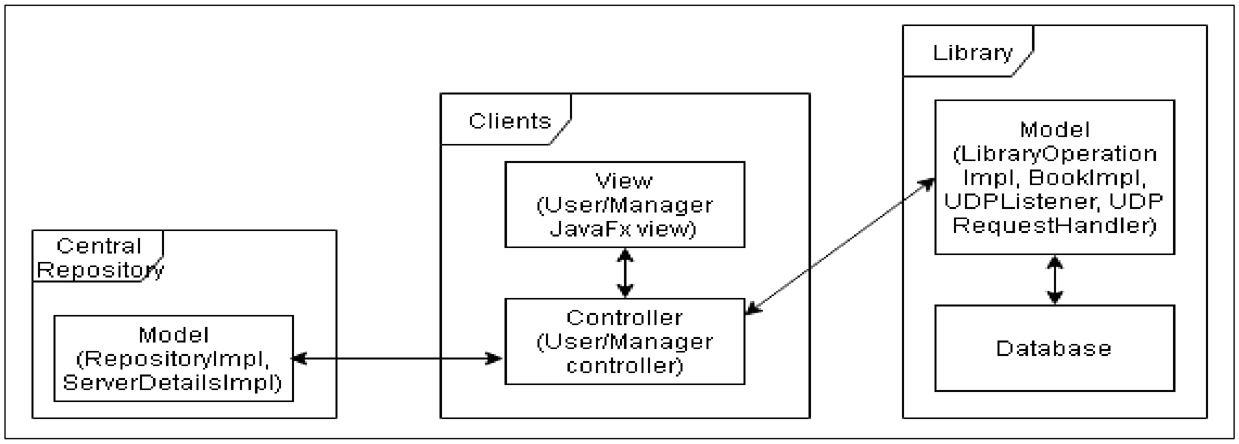

Library
Model (LibraryOperation Impl, BookImpl, UDPListener, UDP RequestHandler)
Database
Clients
View (User/Manager JavaFx view)
Controller (User/Manager controller)
Central Repository
Model (RepositoryImpl, ServerDetailsImpl)

Figure 3. Application Architecture Diagram

Class diagram

The most complicated component, in terms of functionality and implementation, of our distributed application are three libraries which are replicas of each other in terms of functionality they provide and business logic implementation. But the only difference is that they are configured and uses different ports for UDP connection and have different individual databases. So the dependencies and functionalities of various classes are shown in class diagram, in *Figure3*, which represent Concordia Library.

Class Description:

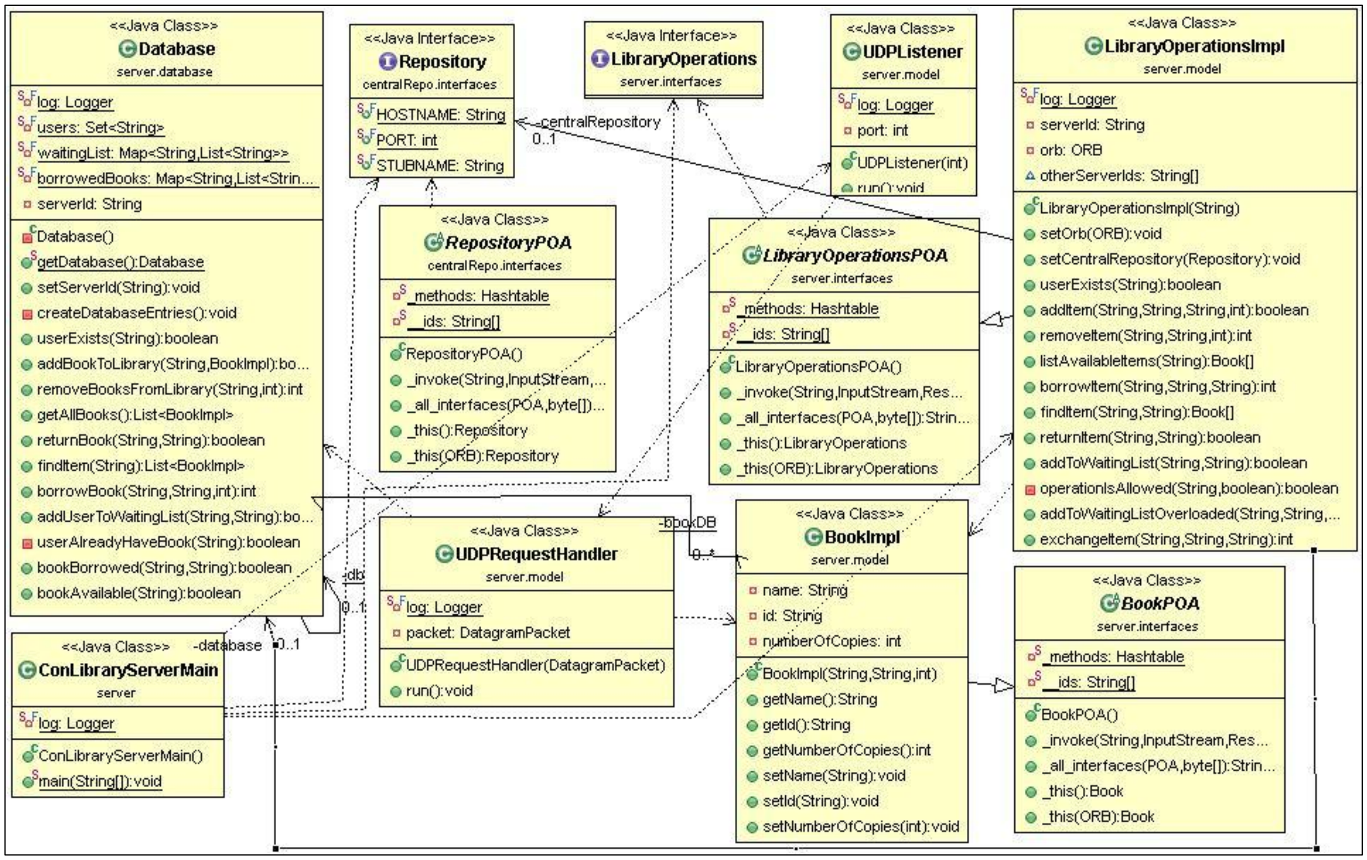
ConLibraryServerMain: Just like in Assignment 1, this is the entry point of library server component and there is not major change functionality wise apart from the fact that now it used CORBA classes to register its detail with Central Repository.

LibraryOperationsPOA: This is an abstract class for library operations generated by compiling IDL file, which is implemented by all the library servers in LibraryOperationsImpl java class to provide the business logic to methods defined in LibraryOperations IDL file and interface. So that those methods can be called by client classes using library opetations stubs fetched from ORB.

LibraryOperationImpl: This class is the extends of LibraryOperationsPOA abstract class and provides business logic to those methods which are defined in LibraryOperations interface in IDL. This class provide total 10 operations out of which 7 are as per assignment requirements which are Add Item, Remove Item, List Item, Borrow Item, Find Item, Return Item, exchange Item. Additional 3 operations are to check if given user exists or not and to add a user to waiting list and its overloaded version.

BookPOA: Similar to LibraryOperationsPOA, this is another abstract class holding CORBA related information and need to be extended by BookImpl class.

BookImpl: This class is the implementation of Book interface. This is a POJO class which represent a book in application.



<<Java Class>>
Database
server.database
log: Logger
users: Set<String>
waitingList: Map<String,List<String>>
borrowedBooks: Map<String,List<Strin...
serverId: String
Database()
getDatabase():Database
setServerId(String):void
createDatabaseEntries():void
userExists(String):boolean
addBookToLibrary(String,BookImpl):bo...
removeBooksFromLibrary(String,int):int
getAllBooks():List<BookImpl>
returnBook(String,String):boolean
findItem(String):List<BookImpl>
borrowBook(String,String,int):int
addUserToWaitingList(String,String):bo...
userAlreadyHaveBook(String):boolean
bookBorrowed(String,String):boolean
bookAvailable(String):boolean
<<Java Interface>>
Repository
centralRepo.interfaces
HOSTNAME: String
PORT: int
STUBNAME: String
-centralRepository
0..1
<<Java Interface>>
LibraryOperations
server.interfaces
<<Java Class>>
UDPListener
server.model
log: Logger
port: int
UDPListener(int)
run():void
<<Java Class>>
LibraryOperationsImpl
server.model
log: Logger
serverId: String
orb: ORB
otherServerIds: String[]
LibraryOperationsImpl(String)
setOrb(ORB):void
setCentralRepository(Repository):void
userExists(String):boolean
addItem(String,String,String,int):boolean
removeItem(String,String,int):int
listAvailableItems(String):Book[]
borrowItem(String,String,String):int
findItem(String,String):Book[]
returnItem(String,String):boolean
addToWaitingList(String,String):boolean
operationIsAllowed(String,boolean):boolean
addToWaitingListOverloaded(String,String,...
exchangeItem(String,String,String):int
<<Java Class>>
RepositoryPOA
centralRepo.interfaces
_methods: Hashtable
__ids: String[]
RepositoryPOA()
_invoke(String,InputStream,...
_all_interfaces(POA,byte[])...
_this():Repository
_this(ORB):Repository
<<Java Class>>
LibraryOperationsPOA
server.interfaces
_methods: Hashtable
__ids: String[]
LibraryOperationsPOA()
_invoke(String,InputStream,Res...
_all_interfaces(POA,byte[]):Strin...
_this():LibraryOperations
_this(ORB):LibraryOperations
<<Java Class>>
UDPRequestHandler
server.model
log: Logger
packet: DatagramPacket
UDPRequestHandler(DatagramPacket)
run():void
-bookDB
0..*
-db
0..1
<<Java Class>>
BookImpl
server.model
name: String
id: String
numberOfCopies: int
BookImpl(String,String,int)
getName():String
getId():String
getNumberOfCopies():int
setName(String):void
setId(String):void
setNumberOfCopies(int):void
<<Java Class>>
BookPOA
server.interfaces
_methods: Hashtable
__ids: String[]
BookPOA()
_invoke(String,InputStream,Res...
_all_interfaces(POA,byte[]):Strin...
_this():Book
_this(ORB):Book
<<Java Class>>
ConLibraryServerMain
server
-database
0..1
log: Logger
ConLibraryServerMain()
main(String[]):void

Figure 4. Class Diagram of library component.

Database: This class holds the application data in various data structures. This act is the in memory database for library. It has a **users: Set<String>** which act as the ids of users registered with this library, it also has a **bookBD: Map<String, Book>** which is used to store the details of books which library have. Each book instance is mapped against a bookId in this map. It has a **waitingList: Map<String,List<String>>** which is used to store all the user waiting for a book in a list of string mapped to that book id. Another data structure is **borrowedBooks: Map<String, List<String>>** which holds a list of users, who borrowed a particular book from library, mapped against the borrowed book id.

This class also provides with various methods to manipulate these data structures in synchronized thread-safe way to support the functionalities offered by library operations.

UDPListener: This class listens to a specific port to handle all the UDP requests made on that port by other libraries. Once it gets a request, it created another thread to handle that request and pass the details of the request to that thread to do processing and return the result back to requesting library.

UDPRequestHandler: This is the class which handle the request made by other libraries and captured by UDPListener thread. Based on the details which are passed by UDPListener to this class, this class do the required processing and handle the request by invoking required methods on database and sending the requesting library response of these database invocation calls.

PseudoCode

This Assignment have an additional functionality added to assignment 1 which is allowing user to exchange the book he already have with one which he want to. This section gives the pseudo code for that functionality.

exchangeItem (studentID, newItemID, oldItemID)

```
• set bookIsBorrowed and bookIsAvailable to false
• if oldItemID is borrowed then set bookIsBorrowed to true
• if newItemID is available then set bookIsAvailable to true
• if(!bookIsBorrowed) return -1  // which indicate that this book is not borrowed by user and nothing can be done further
• else if (!bookIsAvailable) return 0  // which indicate that based on user's choice old book can be returned and user can be added to newItemID's waiting list
• else set bookBorrowSuccessful = false && bookReturnSuccessful = false
•     if(bookIsBorrowed && bookIsAvailable) then bookBorrowSuccessful = borrowBook(studentId, newItemID)
•         If(bookBorrowSuccessful) then bookReturnSuccessful = returnItem(studentId, oldItemID)
•             If(bookReturnSuccessful) then return 1 // which indicated that exchange successful
•             else return 0 // which indicate that based on user's choice old book can be returned and user can be added to newItemID's waiting list
```

Use Cases

As all the functionality of Assignment2 as similar to Assignment 1 so we will be not doing rigorous testing of previous functionality but just testing few cases but for new exchange item functionality we will be checking for various cases.

For testing purpose each server have few users'/managers registers to it and a list of books. List of users registered to each library (with starting 3 character of username being different which is according to library server) is CONM1111, CONM1112, CONM1113, CONM1114, CONM1115, CONU1111, CONU1112, CONU1113, CONU1114, CONU1115, CONU1116, CONU1117, CONU1118, CONU1119, CONU1120.

Similarly books in library database (starting three character of book id is as per the library server id) is as follow CON6231, CON6641, CON6491, CON6651, CON6481 with their corresponding book names as Distributed Systems, Advanced Programming, Systems Software,Algorithm Design, System Requirements Spec.

Method to test: **Borrow Book()**

In this test case we want to test if multiple books can be borrowed by a library user from his own library or not. We make request two times and it should be successful both times.

userID	ItemId	ItemName	Result
CONU1112	CON6641		Book is issued to CONU1111
MONU1112	CON6641		Book is issued to MONU1111

Method to test: **Find Item()**

Given an item name we want to test if user client can find matching books from all libraries or not. Client should show item ids and quantity of books available on all the servers whose names match with the provided name.

userID	ItemId	ItemName	Result
CONU1111		Distributed Systems	CON6231 3, MCG6231 5, MON6231 4

Method to test: **Return Item()**

Case1: User CONU1111 have 1 MON6231 book. We will try to return MON6231 2 times. User should only be able to return book once as he has only 1 MON6231 issues on his name.

userID	ItemId	ItemName	Result
CONU1111	MON6231		Book returned to library successfully.
CONU1111	MON6231		Unable to return book to library

Method to test: **List Item()**

We will provide manager id to manager client and query for listing the items available at the serer. It should return all the items available in the library.

userID	ItemId	ItemName	quantity	Result
CONM1111				CON6231 Distributed Systems 5,CON6641 Advanced Programming 5, CON6491 Systems Software 5,CON6651 Algorithm Design 5, CON6481 System Requirements Spec 5

Method to test: **Add Item()**

We will add an item with correct item id and the server should add the item to its database.

userID	ItemId	ItemName	quantity	Result
MCGM1111	MCG1234	Temp book	2	Book is added to library.

Method to test: **Remove Item()**

First we will try to delete newly added item more than the quantity in which its available and server should not delete anything but just let user know that it can't perform operation and lastly we will try to delete all the copies of this item and server should be able to perform this operation and inform the manager through client.

userID	ItemId	ItemName	quantity	Result
MCGM1111	MCG1234	Temp book	3	Can't delete more books than library currently have.
MCGM1111	MCG1234	Temp book	-1	Item is completely deleted from library.

Method to test: **Exchange Item()**

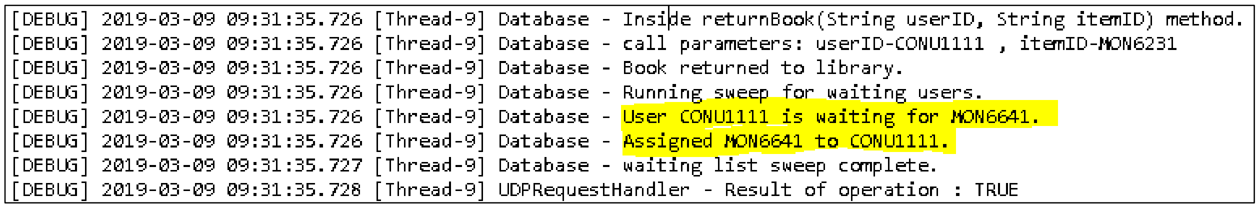
Case1: First with user CONU1111 we borrow book CON6231 and then exchange it with MON6231.

Case2: Then we again try to borrow book CON6231 and exchange it with another book from same library say MON6641, in this case as user already have a book, MON6231, library should let us exchange the book but asked us to put us to waiting list for MON6641 and return CON6231.

userID	NewItemId	OldItemId	Result
CONU1111	MON6231	CON6231	Book exchange is successful.CONU1111
CON1111	MON6641	CON6231	Returned CON6231and added CONU1111 to waiting list of MON6641 book.

Case3: Now if we try to return the MON6231 then user should automatically get MON6641 assigned to him.

Server logs for this transaction are as follows:



```
[DEBUG] 2019-03-09 09:31:35.726 [Thread-9] Database - Inside returnBook(String userID, String itemID) method.
[DEBUG] 2019-03-09 09:31:35.726 [Thread-9] Database - call parameters: userID-CONU1111 , itemID-MON6231
[DEBUG] 2019-03-09 09:31:35.726 [Thread-9] Database - Book returned to library.
[DEBUG] 2019-03-09 09:31:35.726 [Thread-9] Database - Running sweep for waiting users.
[DEBUG] 2019-03-09 09:31:35.726 [Thread-9] Database - User CONU1111 is waiting for MON6641.
[DEBUG] 2019-03-09 09:31:35.726 [Thread-9] Database - Assigned MON6641 to CONU1111.
[DEBUG] 2019-03-09 09:31:35.727 [Thread-9] Database - waiting list sweep complete.
[DEBUG] 2019-03-09 09:31:35.728 [Thread-9] UDPRequestHandler - Result of operation : TRUE
```

Case 4: CONU1111 borrow book MON6641 and will try to exchange it with MCG6231. Ideally user should be able to do that.

userID	NewItemId	OldItemId	Result
CONU1111	MCG6231	MON6641	Book exchange is successful.CONU1111