

Homework 3, RMI And Databases

Network Programming, ID1212

Vasileios Charalampidis, vascha@kth.se

December 03, 2017

1 Introduction

The purpose of this third assignment is to develop a distributed application, which uses remote method invocation for inter-process communication and persists data in a database.

To meet these goals, a Java application was deployed to handle the case of a file catalog. In this application, a client first registers to the system (if he hasn't done it already) by giving a username and a password. The username must be unique, otherwise a warning message from the server is displayed to the user. If everything goes well with the registration, the client is now able to log in by providing the correct username and password. Again warning messages are displayed in case of wrong input. After the successful log in, the client is now able to continue with the following commands:

- upload <file path><access method><permissions>

for uploading a file, where a) the file path is the one extracted by Windows OS e.g. "C:\Users\chara\Documents\test.c" (might not work in different OS, as a processing of this path took place), b) the access method can either be private (for personal usage only) or public (a file visible to anyone), c) if the access is public, then the user must also provide the third parameter for permissions i.e. write (to be able for anyone to change or delete the file) or read (to just give the permission for the public to retrieve the file). An example of this command can be:

upload "C:\Users\chara\Documents\test.c" public write

- download <file name>

for downloading a file, where the user must provide the name of a file available in the server's database. Private files are not visible to the public, but only to the owner for downloading. An example of this command can be:

download test.c

- list
for listing all the available files (found in server's database) for the corresponding user, by including information about size, owner, type of access and permission.
- delete <file name>
for deleting a file in server's database (only with the right permissions i.e. public file with write permission if the current client is not the owner of the file), where the file name is the name of the file that the client wants to delete. An example of this command can be:
delete test.c
- status <file name>
for getting username information i.e. retrieves, updates, deletes, on specific public files that the corresponding user owns, where the file name is the name of the public file that the user is interested in. An example of this command can be:
status test.c
- unregister
for unregistering from the system.
- logout
for logging out from the server.
- quit
for quitting from the application run.

The deployed program meets some specific requirements, so as to present an acceptable solution for this assignment. The first requirement is that client and server communicate only using remote method invocation. The exception is file transfer, which is performed using TCP sockets. The second requirement is that only the server is allowed to register itself in an RMI registry. When the server makes a call to a client, it must be via a remote reference passed to the server by the client. The third requirement is that the server uses a database to keep records on each user (user name and password) and on each file in the catalog (name, size, owner, public/private access permission, write/read permissions). The fourth requirement is that The clients do not store any data. All data entered by a user is sent to the server for processing, and all data displayed to a user is received from the server. Here, Data, means only username, password and file information (name, size, owner, public/ private access permission, write/read permissions, and content). Other parts of the user interface, like instructions to the user, are best generated by the client. The last requirement is that the user interface must be informative. This means that the current state of the program must be clear to the user, and the user must understand what to do next.

2 Literature Study

In order to accomplish this assignment, it was very important at the beginning to read carefully the application scenario and all its requirements. After the realization of the given context, the given material i.e. code and videos, at canvas for this course, was more than helpful for the java implementation. The videos were very informative, as the instructor explained in good detail the concepts that were necessary for the implementation i.e. RMI, database access, Java Persistence API. The code of the bank-program was close to File Catalog program, with the difference that now also TCP file transferring occurred. Except from the provided course material, information about JPA found on the web was also part of dealing with the application.

What became clear from the materials used, was that RMI can save us a lot of work when writing a distributed application that provides remote communication between Java programs. And that because, now we don't care about ports and hosts, but rather for a specific registry. RMI is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM. As for Java Persistence API, it refers to a collection of classes and methods to persistently store data into a database which is provided by the Oracle Corporation. JPA has an architecture with specific steps that must be taken in order, so that to avoid any errors, and finally store data correctly to database. The first step in JPA is associated with the persistence.xml file, which defines the provider, the classes and properties of the persistence unit of our program.

3 Method

The Java implementation was created with "IntelliJ IDEA" by JetBrains. It is a Java integrated development environment (IDE) for developing computer software, with focus on Java applications. It is a convenient tool for the user as it returns code predictions, snippets, error highlighting, and many more.

The database used in the assignment was Apache Derby, which is an open source relational database implemented entirely in Java.

Before proceeding with the actual file catalog, some tests were performed to smaller units e.g. a simple TCP socket program for sending and receiving files, data exchanged with RMI calls etc. so that to be sure that specific concepts will work in the final implementation.

To evaluate that the final solution met the requested requirements, several manual acceptance tests were performed to the code e.g. upload files, download files, login, register, etc. Also, appropriate system messages were introduced as a way for following up the program flow.

4 Result

A layered architecture was used for this assignment (Figure 1). There are three main packages:

- The client package, which contains a) the startup package, that includes the Main method, for starting a client while looking up for the remote object associated with our implementation (line 19), b) the view package, that includes the ClientUI class for handling all the user interface of a new client, c) the controller package, that includes the controller class, for connecting the UI with the net package for handling file transferring. The class introduces separate threads for calling the remote processes (line 28 and line 62) in order to prepare server for listening sockets, as well as to avoid client system stacking, d) the net package, that includes the ServerConnection class for handling the communication with TCP sockets for file exchanging with the server (line 21 and line 51). All the files that are downloaded from the server are directed to the file ClientDirectory (folder), which is located inside the src folder.
- The common package, which contains: a) the FilesCatalog interface that extends Remote (line 14) for specifying the file's catalog remote methods e.g. register, login, delete, etc, b) the ProcessInput that implements Serializable for server processing the input of the client and returning just the results (more details later on in one of the requirements of the assignment), c) the Commands enumerator for all the possible command types.
- The server package, which contains: a) the startup package, that includes the Server class, for starting the server, while creating a remote object registry (line 31) that accepts calls on a specific port, b) the net package, that includes the FileTransfer class for handling the TCP socket communication with the client for file transferring. The files that are uploaded to the server are saved in the ServerDirectory (folder) inside the src folder, c) the controller package, that includes the Controller method for the implementation of all the file's catalog remote methods. This is the only server class that can be called remotely, d) the model package, which is divided further to clientHandling package and fileHandling package. The first one refers to client oriented operations and includes the ClientsInputProcess class for the input processing of the corresponding client; the ClientsSystem for handling client operations e.g. register, unregister, login etc.; the ClientHandler that handles information for one client e.g. username, password, clientId, and also includes queries and other JPA annotations that will be used from the integration package; the ClientException class for considering client oriented errors. The fileHandling package includes the FileSystem class for handling all the file system operations e.g. delete, listFiles, etc.; the FileEntity class that handles one file entity and includes queries that will be used from integration package; the StatusInfo class for file information purposes i.e. file retrieves, updates, deletes;

the `FileException` class for considering file oriented errors, and finally e) the integration package, that includes the `FileSystemDAO` class. This data access object (DAO) encapsulates all database calls in the file catalog application.

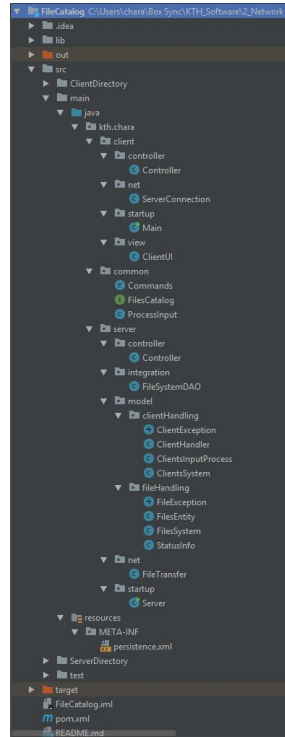


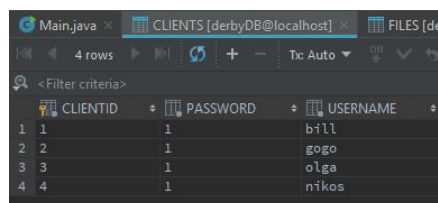
Figure 1: The architecture of the file catalog program

Considering now each requirement separately:

- The client and server must communicate only with RMI, with the only exception the file transferring for uploading and downloading, which must be handled with TCP sockets. This was achieved by introducing the `FileCatalog` interface in the common package. The `ClientUI` class performs calls to this interface by passing arguments (like [line 178](#)). The server's controller package that includes the controller class extends `UnicastRemoteObject` and implements `FileCatalog` interface ([line 22](#)) for implementing all the file's catalog remote methods. As for the TCP communication, this was handled by introducing `net` packages to both client and server. When the client wants for example to upload a file, the client's controller class is called ([line 155](#)) to prepare the server with a new thread ([line 30](#)) so that to open a listening socket. Then again the controller is called ([line 157](#)) to send the file path and the file name to the `ServerConnection` class ([line 47](#)), which actually sends the file with a TCP socket ([line 21](#)). From the server side now, firstly an

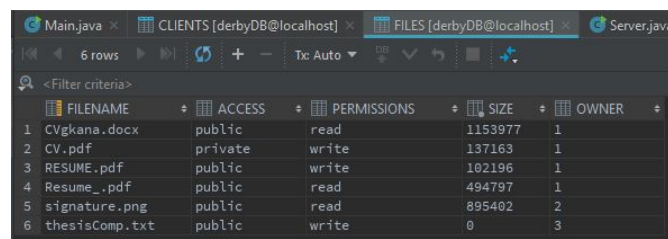
upload RMI call was introduced (line 32) to prepare the server to open a listening socket, and execute as result the FileTransfer class in net package (line 20).

- Only the server should register to RMI registry. This is clear in the Server class in server's startup package (line 35). If the registry is not yet in the list, then it is created while calling the method rebind to connect to the registry the Controller method (line 38).
- The server uses a database to keep records on each client and on each file in the catalog. This was achieved by introducing a persistence unit in the implementation. The persistence unit consists of two entities, the ClientHandler (line 21), which refers to the database table that includes the clientID, username and password of clients (Figure 2); and the FileEntity (line 26), which refers to the database table that includes the filename, size, owner (here it is proper to mention that there is a ManyToOne association, line 70, between files and a unique owner), type of access and type of permission (Figure 3). This persistence unit is connected with a Derby database (line 8). All the database operations e.g. delete a file entity (line 127) are performed in the FileSystemDAO class in server's integration package.



	CLIENTID	PASSWORD	USERNAME
1	1	1	bill
2	2	1	gogo
3	3	1	olga
4	4	1	nikos

Figure 2: The database table for the clients



	FILENAME	ACCESS	PERMISSIONS	SIZE	OWNER
1	CVgkana.docx	public	read	1153977	1
2	CV.pdf	private	write	137163	1
3	RESUME.pdf	public	write	102196	1
4	Resume_.pdf	public	read	494797	1
5	signature.png	public	read	895402	2
6	thesisComp.txt	public	write	0	3

Figure 3: The database table for the files

- The clients do not store any data. This was achieved in four sections. The first one has to do with the input commands in the console from the client. By introducing the class ProcessInput, in common package, that implements Serializable (line 13), the client received all the appropriate processed feedback concerning his actions by the server. The client just makes a call to the processInput method

in FilesCatalog interface ([line 29](#)), in order for the server to process the client's commands. The second section has to do with displaying the available file list with information regarding the available files in the database ([line 110](#)). A string is returned from the server with all the corresponding information ([Figure 4](#)). The third section has to do with the file information regarding one unique file e.g. retrieves, updates, deletes from specific usernames ([line 121](#)). The server returns a string with all the appropriate information ([Figure 5](#)). The last section refers to all the exceptions regarding client and file errors. For that purpose, two separate classes (ClientException and FileException in server's model package), which both extends Exception method, were introduced for returning appropriate error messages.

```
"C:\Program Files\Java\jdk-9.0.1\bin\java" ...

Welcome to the File Catalog!
The commands of the system are:

register <username> <password>           //create a new account//
login <username> <password>              //login to the system//
quit                                     //to quit the system//

> login bill j

Hi bill!
Available Commands:

upload <path> <private>                 //to upload a private file//
upload <path> <public> <write or read>   //to upload a public file//
download <filename>                     //to download a file//
list                                    //to view all the available files//
delete <filename>                       //to delete a specific file//
status <filename>                      //to check the status of a specific public file//
unregister                              //to unregister your account//
logout                                  //to logout from catalog//
quit                                    //to exit the system//

> list
File: CVgkana.docx, size: 1153977 bytes, owner: bill, access: public, permission: read
File: CV.pdf, size: 137163 bytes, owner: bill, access: private, permission: write
File: RESUME.pdf, size: 102196 bytes, owner: bill, access: public, permission: write
File: Resume_.pdf, size: 494797 bytes, owner: bill, access: public, permission: read
File: signature.png, size: 895402 bytes, owner: gogo, access: public, permission: read
File: thesisComp.txt, size: 0 bytes, owner: olga, access: public, permission: write
└
```

Figure 4: The list with file information

- The user interface must be informative. A responsive UI was introduced ([line 15](#)) with all the appropriate information for the client to understand what to do next ([Figure 6](#)).

The final code of the Java application can be found [here](#). Several comments are included there discussing about essential parts of the program.

5 Discussion

The main goals of the assignment were to learn about remote method invocation for inter-process communication and persisting data in a database.

```
> login bill j
Hi bill!
Available Commands:

upload <path> <private>                //to upload a private file//
upload <path> <public> <write or read> //to upload a public file//
download <filename>                    //to download a file//
list                                  //to view all the available files//
delete <filename>                     //to delete a specific file//
status <filename>                     //to check the status of a specific public file//
unregister                             //to unregister your account//
logout                                //to logout from catalog//
quit                                  //to exit the system//

> list
File: CVgkana.docx, size: 1153977 bytes, owner: bill, access: public, permission: read
File: CV.pdf, size: 137163 bytes, owner: bill, access: private, permission: write
File: RESUME.pdf, size: 102196 bytes, owner: bill, access: public, permission: write
File: Resume_.pdf, size: 494797 bytes, owner: bill, access: public, permission: read
File: signature.png, size: 895402 bytes, owner: gogo, access: public, permission: read
File: thesisComp.txt, size: 0 bytes, owner: olga, access: public, permission: write

> status CVgkana.docx
Retrieved by: [gogo, olga], updated by: none, deleted by: none
|
```

Figure 5: The specific usernames regarding operations on a public file

```
"C:\Program Files\Java\jdk-9.0.1\bin\java" ...

Welcome to the File Catalog!
The commands of the system are:

register <username> <password>          //create a new account//
login <username> <password>             //login to the system//
quit                                    //to quit the system//

> register john 123
Registration success! Login to continue!

> login john 123
Hi john!
Available Commands:

upload <path> <private>                //to upload a private file//
upload <path> <public> <write or read> //to upload a public file//
download <filename>                    //to download a file//
list                                  //to view all the available files//
delete <filename>                     //to delete a specific file//
status <filename>                     //to check the status of a specific public file//
unregister                             //to unregister your account//
logout                                //to logout from catalog//
quit                                  //to exit the system//

|
```

Figure 6: The informative UI

The requirements of the presented code can be summarized in using only RMI between client and server communication (except from file transfer that was handled with TCP sockets), registering only the server to RMI registry, the server using a database for keeping records on each user and on each file in catalog, the client not storing any data, and finally having an informative UI. Any problems that occurred during the execution were immediately identified as many 'throw exception' rules were implemented in the Java code.

If something was to be done differently, that would be again as in the previous two assignments, the testing part. It would have been beneficial to perform unit tests (e.g. [JUnit](#)), so that to guaranty that everything performs fine without any issues.

6 Comments About the Assignment

This specific assignment was a bit more demanding. As a result, more time was needed to accomplish it, which was about four days (with several breaks in between).