Web Programming Task: Request-Reply Messaging App

The purpose of this paper is to familiarize you with network programming. In the context of this work, a distributed messaging system using a simple request-reply protocol will be developed. Clients send the Server a Request and then the Server responds with a Response and their connection is terminated. The technologies you are invited to use are either that of Sockets (Sockets-I/O streams-Threads), or RMI (Remote Method Invocation).

In this system different users will be able to create accounts and send messages to each other, as the server will manage them. This functionality will be provided to them:

a) a server program, which will have the ability to handle multiple requests from clients simultaneously.

b) client programs, each of which will have the ability to send requests to the server.

The main elements of the project are presented below:

A. Message

Every message sent, received or stored on the server must have the following fields:

Property	Description
boolean isRead	Indicates whether the message has already been read.
sender string	The sender of the message.
String receiver	The recipient of the message.

String body	The text of the message.

You can fill in any other property you consider necessary for the implementation of the system.

B. Account

Each user account will be stored on the server with the following Form:

Property	Description
String username	The user name. It consists only of alphanumeric characters and the special character "_".
Int authToken	A unique user identification number (generated by the server and personal/hidden).
List<Message> messageBox	The user's mailbox, which is a list of Messages.

You can fill in any other property you consider necessary for the implementation of the system.

C. MessagingClient

The client program is where the communication between the user and the server is implemented.

The program will accept input from the user by sending it to the server, while simultaneously receiving data from the server and displaying it appropriately to the user.

MessagingServer

The server will run continuously as a service "listening" for incoming requests from clients. Each incoming request will be assigned to a different thread (you can use Threads or the RMI mechanism), in order to allow multiple requests to be serviced simultaneously.
Also, the server will have a list of accounts (Account), so that data such as registered users, their passwords and their mailboxes can be kept. Finally, methods for communicating with the user must be implemented.

Functions

The delivered programs must necessarily perform the functions we will describe below. Executable programs must give results **exactly in the form we describe in the Scenario tables.**

Server

The server should run in the following way:

```
java server <port number>
```

where Port Number is the door at which you will listen to requests.

Client

The Client is the user's interface with the program. The general run command is:

```
java client <ip> <port number> <FN_ID> <args>
```

where

- ip: The IP address of the Server
- port number: The port on which the Server is listening
- FN_ID: The identifier of the function to be executed
- args: the function parameters The

following functions are provided:

Create Account (FN_ID: 1)

`java client <ip> <port number> 1 <username>`

Creates an account for the user and uses the given username.
The function returns a unique token which is used to authenticate the user in subsequent requests.

Script	**Print**
Success	<integer>.

Example:
`$>java client localhost 5000 1 tester`
`1024`
In the example above, the tester user will, from now on, use the number `1024` as the auth Token in subsequent requests.	

The user already exists	Sorry, the user already exists
Example:	

`$>java client localhost 5000 1 tester` `Sorry, the user already exists`

Incorrect username format	Invalid Username

`Example`
`$>java client localhost 5000 1 inv@-lid`
`Invalid Username`	

Show Accounts (FN_ID: 2)

`java client <ip> <port number> 2 <authToken>`

Shows a list of all accounts in the system.

Script	**Print**
In all cases it prints the list of users in the format given	1. <username_1> 2. <username_2> … n. <username_n>

Example:
`$>java client localhost 5000 2 1024`
`1. demo`
`2. raven_13`
`3. dr4g0n_sl4y3r`	

Send Message (FN_ID: 3)

`java client <ip> <port number> 3 <authToken> <recipient> <message_body>.`

Sends a message (<message_body>) to the account with username <recipient>.

Script	**Print**
Successful mission	OK

Example:
`$>java client localhost 5000 3 1024 tester "HELLO WORLD"`
`OK`	

If the profile of the <recipient> user there is no	User does not exist

Example:
`$>java client localhost 5000 3 1024 friend "HELLO WORLD"`
`User does not exist`	

Show Inbox (FN_ID: 4)

java client <ip> <port number> 4 <authToken>

Displays the list of all messages for a specific user.

Shows a list of all messages in the user's messagebox. H
The format that must be printed is as follows:

Script	Print
Success	<message_id_1>. from:<username_x>. *? <message_id_2>. from:<username_y>. *? … <message_id_n>. from:<username_z>. *?

The asterisk will only be visible if the message has not been read. Example:
`$>java client localhost 5000 4 1024`
`27. from: raven_13*`
`43. from: demo*`
`55. from: raven_13*`
`58. from: raven_13*`
`67. from: dr4g0n_sl4y3r`
 	
In our example, the message of user `dr4g0n_sl4y3r` has already been read.	

ReadMessage (FN_ID: 5)

java client <ip> <port number> 5 <authToken> <message_id>

This function displays the content of a user message with id <message_id>. The message is then marked as read. If the

message exists, the program prints the following

Script	Print
Success	(<sender>) <message>

Example:
`$>java client localhost 5000 5 1024 43`
`(demo)Hello World`	

The message does not exist	Message ID does not exist

Example:
`$>java client localhost 5000 5 1024 1111`
`Message ID does not exist`

DeleteMessage (FN_ID: 6)

`java client <ip> <port number> 6 <authToken> <message_id>`

This function deletes the message with id <message_id>.

Script	**Print**
Success	OK

Example:
`$>java client localhost 5000 5 1024 43`
`OK`	

The message does not exist	Message does not exist

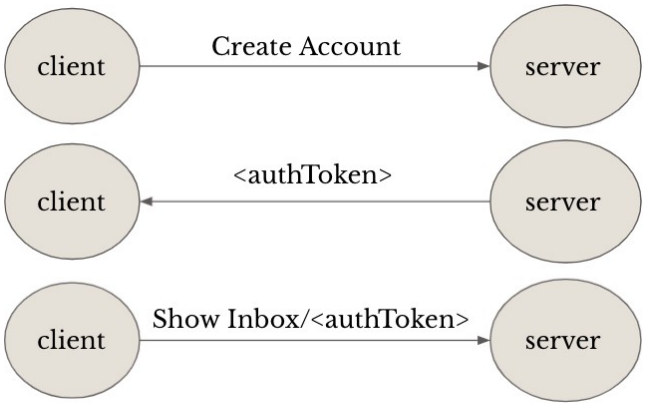
Example:
`$>java client localhost 5000 5 1024 1111`
`Message does not exist`	

auth token

More generally, since the communication protocol that will be created does not keep user information in each session, we need to ensure that:

- no user can read another user's data.
- no user can send messages as another user.

This is why we use the Auth Token in every Request. The Server must associate each auth token with an Account (when creating the account).



In case of a wrong Auth Token (i.e. the authToken is not assigned to a user) all functions must return the corresponding message.

Script	Print
AuthToken error	*Invalid Auth Token*

Deliverables

Deliver a zip file with Latin characters and format:

<aem>_<name>_<surname>.zip

e.g. 1243_petros_riginos.zip

containing the following:

- src/: source code files.
- jars/:
  - Client.jar: Client runtime file
  - Server.jar: Server runtime file
- README.md A text file in which you briefly describe the classes you implemented and the assumptions you made about the system implementation.

Attention: it is important that the above are **strictly submitted in the correct format and file structure** in order to complete the submission process smoothly. We recommend that you run your projects as jar files to confirm their proper function before submitting them.

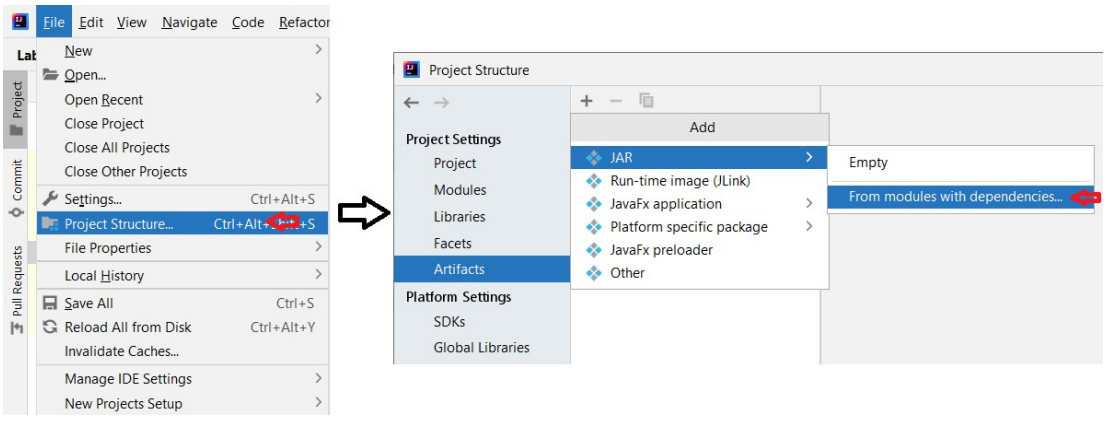
```
java -jar Server.jar 5000
java -jar Client.jar localhost 5000 1 demousername
```

Create Executable Jar (IntelliJ):

Step 1:

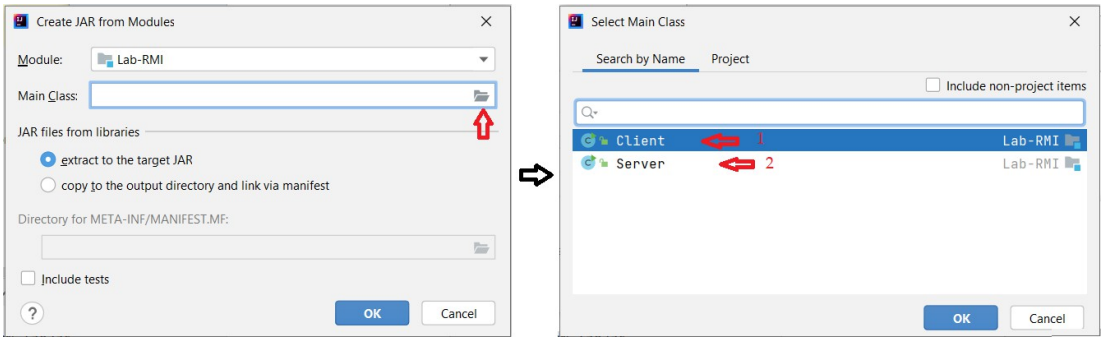
We follow the route...

File > Project Structure > Project Settings > Artifacts > Click green plus sign > Jar > From modules with dependencies...

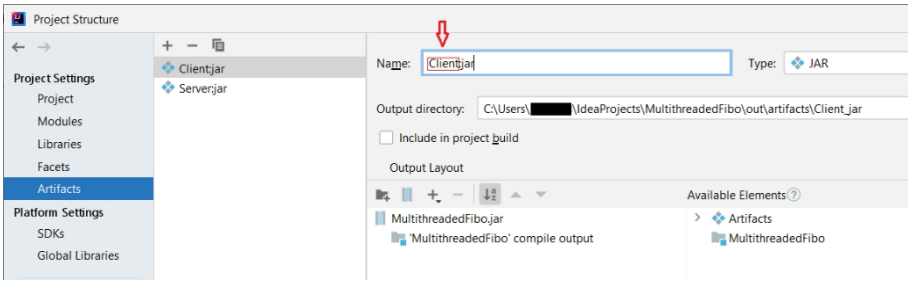


Step 2:

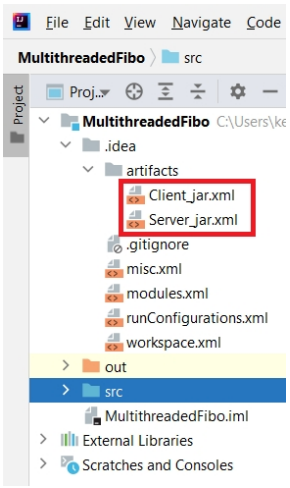
Suppose we have a project with two files that carry the main method, this means we need to create two .jar files. The second image below shows these two project files i.e. Client.java and Server.java. Arrows 1 and 2 imply that we need to repeat the process of creating the .jar files for both.



For each .jar file creation process we should give the appropriate title to the file we are creating (e.g. Client or Server) within the red box shown in the image below, since by default, Intellij gives both the same project title. We do this so that we know at the end in which folders the final .jar files will be generated for each.

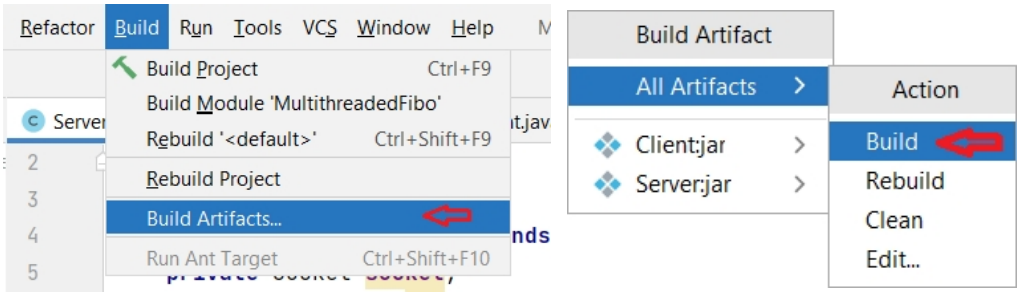


Then two xml files will appear in the project navigator as shown in the following picture in the red box:



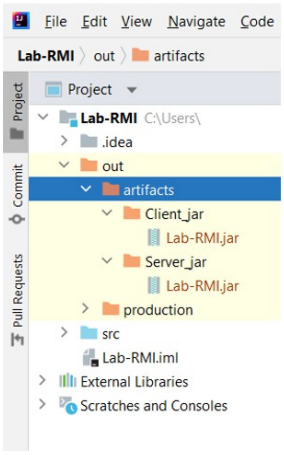
Step 3:

Finally, we create the compressed .jar executables as shown in the image below.

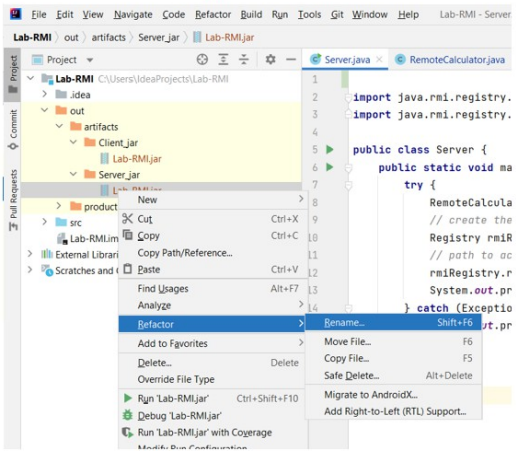


Intellij will save the final two .jars in the out folder of the project inside two sub-folders Client_jar and Server_jar, respectively. Intellij assigns the same name (the project name i.e. <projectName>.jar) to both. However because they will be delivered in a folder called jars -as it asks in the deliverables, you will need to grab them and rename them (right-click > refactor > rename > (proceed anyway)) to Client.jar and Server.jar when you save them in the deliverable folder which will eventually become a zip file.

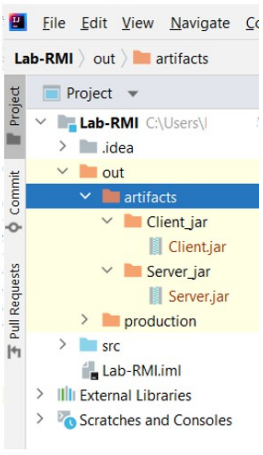
1


File Edit View Navigate Code
Lab-RMI out artifacts
Project
Lab-RMI C:\Users\
.idea
out
artifacts
Client_jar
Lab-RMI.jar
Server_jar
Lab-RMI.jar
production
src
Lab-RMI.iml
External Libraries
Scratches and Consoles
Project
Commit
Pull Requests

2


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Lab-RMI - Server
Lab-RMI out artifacts Server_jar Lab-RMI.jar
Server.java
RemoteCalculator.java
Lab-RMI C:\Users\IdeaProjects\Lab-RMI
.idea
out
artifacts
Client_jar
Lab-RMI.jar
Server_jar
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+F7
Analyze
Refactor
Add to Favorites
Delete... Delete
Override File Type
Run 'Lab-RMI.jar' Ctrl+Shift+F10
Debug 'Lab-RMI.jar'
Run 'Lab-RMI.jar' with Coverage
Rename... Shift+F6
Move File... F6
Copy File... F5
Safe Delete... Alt+Delete
Migrate to AndroidX...
Add Right-to-Left (RTL) Support...
import java.rmi.registry.
import java.rmi.registry.
public class Server {
public static void ma
try {
RemoteCalcula
// create the
Registry rmiR
// path to ac
rmiRegistry.r
System.out.pr
} catch (Exceptio
Project
Commit
Pull Requests

3


File Edit View Navigate
Lab-RMI out artifacts
Project
Lab-RMI C:\Users\I
.idea
out
artifacts
Client_jar
Client.jar
Server_jar
Server.jar
production
src
Lab-RMI.iml
External Libraries
Scratches and Consoles
Project
Commit
Pull Requests