Distributed Systems Seminars: 3. Remote Method Invocation

We cannot solve our problems with the same thinking
we used when we created them
Albert Einstein

The Story

Unfortunately, the approach we took previous time failed. And that resulted in us being delayed and you being exhausted and hard-working in order to just have a few bytes transferred. At least, we learned the lesson: those all levels of abstraction are there to decrease the efforts you have to apply in order to develop the new solution you're working on. Today we're going to build our super-amazing-awesome-perfect chat application on the shoulders of giants: using Java Remote Method Invocation platform. As we looked briefly though the description, it appears that RMI allows us to avoid ANY communication burden at all! You just invoke some methods and the magic happens somewhere inside! Isn't that amazing? Let's make this shit sending kittens now!

Today's goal is to try creating a chat client using the Java RMI framework using predefined server interface (defined in DSTSem-03-RMI.zip).

Warning Note

This task is **individual**, thus you have to complete it **on your own**. Whenever identical (or suspiciously similar) solutions are detected, the first one to publish is assumed the author and others get **0 points for the task**. **Note**, that this **does not mean** that you have to be an author of all code in your repository: this is simply not the way how modern IT industry works. You are allowed (**and even encouraged**) to use third-party libraries and solutions from the Internet. However, you should **clearly indicate** where the code originated (it's sufficient to write a comment above the foreign code with URL and the amount of code you borrowed).

This material is **not meant to provide all necessary information** to succeed in the course. It may give you sufficient knowledge to satisfy the minimal criteria, but you should take this document more as a roadmap that simplifies discovering necessary information.

The task will be evaluated manually, thus **provide a meaningful README** file that explains how to run your solution and what to type in order to verify all tasks you solved. **If examiner fails to run your code, you will get 0 points for the task.**

Even though this course **neither focus nor evaluates** the quality of your code, you ought to pay some attention to these properties of your solution: smart design and clean code always pay off on their own.

The Task

1. Create a branch "Seminar03" from your "master", "Seminar01", or "Seminar02" branch and perform the following tasks in this branch **(1 point)**;
2. Retrieve the RMI Registry from lv.rst.uk.to:152 and obtain the proxy object from the remote registry (**+1 point**);
3. Implement "ping", "echo" and "exit" commands. (**+1 point**);
4. Implement "login" and "list" commands. (**+1 point**);
5. Implement message and file sending ("msg" and "file" commands) and receiving. **(+1 point**);
6. **BONUS**: Implement live user list updates (i.e., printing when user logins or logs out) (**+1 point**);
7. **BONUS**: Be the first to finish the 5 main tasks and send the e-mail indicating that to orest.a.lavriv@lpnu.ua **(+1 point**);

Implementation Hints

This time it should be much easier to solve the task. Thus, there will be fewer hints over here. Try solving all tasks, as they're almost of an equal complexity! Continue reading these hints and you should be able to develop the nice and working solution quite easy. The best way to work is solve tasks in the declared order (from 1 to 7) and to read a section assigned to the particular task prior starting any coding.

2. Retrieving an RMI proxy object

Even though RMI usage is quite simple, there are a few things you should be aware of before designing your solution or starting writing any code. First of all, you should learn that RMI in Java is represented in a form of "registries" that contain named remote objects. Whenever server declares RMI usage, it has to create a registry and define its remote objects (the objects that should be accessible over the network) in this registry. Each registry, from the client, is associated to some hostname and a port. More or less, you can understand an RMI registry as a dictionary (key-value collection) of remote objects. You can obtain the remote registry using the following (or similar) command:

```
Registry registry = LocateRegistry.getRegistry(hostname, port);
```

…where the hostname and port specify the address of the remote repository.

After you successfully obtain a registry, you should locate the remote object you're interested in. In order to do that, you should first include `IServer` interface and all related classes into your java project. You can do that by extracting the `DSTSem-03-RMI.zip` into the folder with your code (the folder where is your `pom.xml` file) and refreshing the project in Eclipse (by clicking F5). After you do that correctly, you should see `lpi.server.rmi.IServer.java` file in your package explorer. Afterwards, you can obtain a proxy object with the following command:

```
IServer proxy = (IServer)registry.lookup(IServer.RMI_SERVER_NAME);
```

Afterwards you should be able to invoke any methods declared in `IServer` using this proxy object. Open `IServer.java` file and look through the code and comments over there in order to understand what is available to you now. Finally, if you want (and, actually, you should) to know more about Java RMI, consider reading appropriate manual and some articles describing the details of Java RMI implementation.

3. Implementing "ping", "echo", and "exit" commands

This time implementation of these all commands is pretty easy: you just have to invoke the method and handle possible exceptions properly:

```
try {
   …
   response = proxy.echo(message);
   …
} catch(RemoteException ex){
   // handle communication exception
}
```

4. Implement "login" and "list" commands.

The idea behind these commands is pretty similar to the ones you implemented before, with the only exception that you have a few more exceptions to handle and a session id to store somewhere in between the calls. Session id – is a unique identifier assigned to you by a server that identifies you in all the following calls. This allows you proving server that you are the right user and avoid sending your login and password in every method invocation.

Note, that server cleans up unused sessions from time to time (every minute), and if you did not invoke any methods using some session id over the last minute, your session may expire and you will have to login again in order to continue using the server. This should not be a problem once you implement a regular polling for messages and files, but it's good to be aware of this peculiarity of server's behavior.

5. Implement sending and receiving messages and files.

Sending a message or file is pretty similar to retrieving a list of users, just that you have to provide another object to the server along with the session id. Check the available constructors and methods declared in `lpi.server.rmi.IServer.Message` and `lpi.server.rmi.IServer.FileInfo`, they should simplify some tasks for you. ☺

Receiving, though, is a bit more complex task. Luckily, this time you don't have to take care of any synchronization issues: this time Java RMI framework takes care of that for you. Therefore, the easiest approach you should follow is to create a Timer object once you login and ensure you close it whenever you disconnect or shutdown an application. Afterwards, you should schedule a periodic task that calls appropriate methods on server proxy in order to figure out if there are any new messages or files waiting for you to receive. Here or here you can see some simple examples on how to create your own timer tasks.

6. BONUS Tasks

Tracking a list of active users is more or less identical to monitoring for a new messages and files waiting for you, with a little bit of complexity afterwards. The simplest approach to handle this task is to store the retrieved list of active users somewhere and periodically retrieve a new list and search for new or removed users in comparison to the list you got in your previous invocation.

Evaluation Process

This section briefly indicates how your task will be evaluated.

1. There is the "Seminar03" branch in your repository: **+1 point**;
2. Application in branch "Seminar03" runs successfully if user is following instructions defined in README file and successfully starts when the server is running and fails with exception or prints a connection error message when the server is not available: **+1 point**;
3. Application allows entering ping and echo <any text> commands. Both commands work properly when the server is running and fail when server is unexpectedly turned off. Also, both commands print server response (if any). Finally, application gracefully shuts down when a user types "exit": **+1 point**;
4. Application allows logging in (and properly handles login error messages) and allows listing currently logged in users: **+1 point**;
5. Application allows sending a messages or files to another user (using "msg" or "file" command) and is able to receive a message/file from the remote user (when the user is logged in). Refuses to invoke commands if the user is not logged in: **+1 point**;
6. Application prints the message when a new user logs in to the server and logs out of the server: **+1 point**;
7. You are the first one to send the e-mail indicating that you solved all 5 tasks and indeed all the tasks are working properly: **+1 point**;
   Total: 7 points.