Software architecture

Week 10 - Asynchronous methods, Web sockets and JMS

Requirements

1. Netbeans 11
2. Java Development Kit 8
3. Payara Server

Asynchronous methods - Future Exercise

1. Create a new Java application.

2. Create a class called **FactorialFuture**. Add followings:
   2.1. A private instance variable called **executor** and initialize it with **Executors.newSingleThreadExecutor()**
   2.2. A public method called **calculateFactorial**, which takes a parameter called **number** (is type of Integer). The method returns a value which is a type of **Future<Integer>**.
   2.3. The body of the method should be as follows:

```
return executor.submit(() -> {
    Thread.sleep(1000);
    int fact = 1;
    for (int i = 1; i <= number; i++) {
        fact *= i;
    }
    return fact;
});
```

3. In the main method, create a **FactorialFuture** object. Declare at least 2 **Future<Integer>** variables and initialize them by calling **calculateFactorial** method of **FactorialFuture** class.

4. Implement the following simple algorithm.

```
f1,f2 <- Future<Integer>
while f1 and f2 are not done
    if f1.isDone then print "F1 is done"
    else print "F1 is not done"
    if f2.isDone then print "F2 is done"
    else print "F2 is not done"
    Thread.sleep(300) // add this line as it is
```

```
result1 <- f1.get
result2 <- f2.get
print result1 and result2
System.exit(0) // add this line as it is
```

5. Import statements should be as follows: 5.1. In class **FactorialFuture**

- java.util.concurrent.ExecutorService
- java.util.concurrent.Executors
- java.util.concurrent.Future 5.2. In the **MainClass**
- java.util.concurrent.ExecutionException
- java.util.concurrent.Future

6. Don't forget to add catch/throw statements when it's needed.

7. RUN your code!

Sockets

In this exercise, we will create two applications that communicate with each other via sockets. One of them will act as a server and the other as a client. The server will contain a list of words and their translation (EN / FR). The client will send a request to the server with a word to translate. The server will check if the word exists in the dictionary, if it exists, it should return the translated word, otherwise it should return a custom message.

Server

The instructions to run the server will be made in the main method of our java application.

To create our project using Netbeans, we will follow the steps below:

1. Open Netbeans
2. Create a New Project (File > New Project > Java with Maven > Java Application)
3. Let's call it "DictionaryServer", click on *Finish*. Your project is ready!
4. Create a new Java class (`Server`)
5. Add a `main` method to the newly created class. Your file should look like this:

```
package com.mycompany.server;

public class Server {
    public static void main(String[] args) {

    }

}
```

In the main method, create a Hashmap containing all the words you want to translate:

```
HashMap<String, String> dico = new HashMap<String, String>();
dico.put("inheritance", "héritage");
dico.put("distributed", "réparti");
```

Now, let's set up our server! The first thing to do is to declare a serverSocket and to define a port:

```
ServerSocket connectionServer = new ServerSocket(4444);
```

A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.
When defining a port number, we should avoid using the well-known ports (Port Number 0 to 1023). More details on well known ports: http://www.meridianoutpost.com/resources/articles/well-known-tcpip-ports.php

When the connection server is declared and assigned with a port number, the server just waits, listening to the socket for a client to make a connection request. In order to accept a connection, we need to define a new client socket and bound it to the same port that the Server Socket:

```
clientSession = connectionServer.accept();
```

At this point, the new Socket object puts the server in direct connection with the client. The next step is to prepare the server to accept input from the client and to send back an output. To do this, we will use a `PrintWriter` to send the message and `BufferedReader` to read the incoming messages.

```
out = new PrintWriter(clientSession.getOutputStream(), true);
in = new BufferedReader(new InputStreamReader(clientSession.getInputStream()));
```

Now we can do the operations and **close the connections**.

```
String word, mot;

            while ((word = in.readLine()) != null) {
                mot = (String) dico.get(word);

                if (mot == null) {
                    mot = "sorry, no translation available for \"" + word + "\"
!";
                }
                out.println(mot);
            }
            out.close();
            in.close();
            connectionServer.close();
            clientSession.close();
```

Client

For the client, we will create a new project by following the same steps as when creating the server. In the main method, we will follow the steps required to connect and to make requests to the server.

The first step is to create a new `Socket` with the information of the server (IP and port number):

```
Socket mySession = new Socket("127.0.0.1", 4444);
```

This constructor only creates a new socket when the server has accepted the connection, otherwise, we will get a connection refused exception. That's the reason why, it's recommended to handle the exceptions by surrounding your code with `try..catch()` instructions. More details here: https://docs.oracle.com/javase/tutorial/essential/exceptions/catch.html
Once the connection is established between the client and the server, we can start the communication by setting up the input and output streams:

```
out = new PrintWriter(mySession.getOutputStream(), true);
in = new BufferedReader(new InputStreamReader(mySession.getInputStream()));
```

The input stream of the client is connected to the output stream of the server, just like the input stream of the server is connected to the output stream of the client.

Now we can make our operations. As a reminder, we have to send an input to the server and wait for a result. This input will be a word (`String`), the server checks if the word exists in its dictionary and returns the translation. We can use the code below to request an input from the user, send it to the server and listen to its response:

```
stdIn = new BufferedReader(new InputStreamReader(System.in));
String fromServer, fromUser;

System.out.println("Go on, ask the dictionary server!");
while (!(fromUser = stdIn.readLine()).equals("quit")) {
    out.println(fromUser);
    fromServer = in.readLine();
    System.out.println("-> " + fromServer);
            }
```

Don't forget to close the opened connections!

```
out.close();
in.close();
stdIn.close();
mySession.close();
```

Additional resources

1. Datagram Socket using UDP: https://github.com/doplab/soar-tp/tree/master/week10/UDP_socket
2. UDP Multicast: https://github.com/doplab/soar-tp/tree/master/week10/UDP_Multicast