B5- Java

B-PAV-560

jCoinche

Client/server solution for playing Coinche

v1.0

jCoinche

Client/server solution for playing Coinche

binary name:	jcoinche-server.jar & jcoinche-client.jar
repository name:	Java_jcoinche_$YEAR
repository rights:	ramassage-tek
language:	Java
group size:	2
compilation:	via Maven or Gradle or other, including package, test

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

The goal of this project is to discover the Java language and its associated development.
For the unit's first project, you must implement a client/server solution in order to play Coinche (a version of the French card game, Belote).

You must create a complete solution composed of a **server** program and and **client** program:

- the server accepts the connection from several clients,
- a client that is connected to the server takes on the role of a card player,
- the server must be able to simultaneously handle several rounds of Coinche (for instance, there could be 5 clients connected, 4 of which are in the middle of playing a round and 1 waiting for a new round to begin).

In accordance with the Java unit, the goal is not to reinvent the wheel, nor write many lines of code.
On the contrary, the goal is to be a Software architect and make a library integration.
Therefore, you must understand how different libraries function and code the business logic in order to complete the project (otherwise called the glue that holds each software component together).

Before starting the project, take the time to analyze and understand how each each component works. In theory we will be talking about the state of the art and POC:

- **State of the Art:** study the different possible solutions and choose the right component, depending on the desired needs,
- **POC or Proof of Concept:** make a quick demonstration program that proves the correct functioning of a component or algorithm.

Your active participation in the Workshop preparation, as well as your attentiond during the activity itself, will be a big help.

Component 1: The build system

Your project must use **Maven** or **Gradle** as build system. Do not implement your own build system with scripts or a Makefile!
The desired rules are:

- **package** in order to build both files **jar**: **jcoinche-server.jar** and **jcoinche-client.jar**
- **test** in order to execute your testsuite on your project
- **cover** in order to execute your testsuite on your project and return the code coverage in % for unit tests

Component 2: The network communication

Gone are the days of creating and opening sockets...
There are numerous libraries that allow you to do the encapsulation of the network communication in order to make your projects sturdier and faster to create.

The goal of this section is to create and maintain the connection between a client and a server.
Also think about multi-client management.
There are libraries with asynchronous functions so that the program doesn't jam up.

You can use **Netty** or any other type of library.

Component 3: The protocol

Now you are able to connect several clients to your server, you'll have them communicate.

It's time for you to implement your own RFC :)
To do so, you need to define the actions that the client must be able to send to the server, and those that the server must be able to send to the clients.

In order to send commands/actions to the client or the server, you have to use a binary format.
However, in order to implement the program's code, it is more pratical to use a representation in the form of a data structure or data class. We're talking about serializing data in order to pass a structure/class format to the binary representation in a buffer.

This step is long and tedious to implement! Find and use a serialization library, like **Protocol Buffers** to automatically do the conversion of data classes in binary buffer.

Component 4: Unit tests

The goal now is to test your server or your client according to your test strategy.
In order to do so, you can use **JUnit**, which is Java's classic component.

Netty offers a good integration with **JUnit**

Thinks about the addition of a library to calculate the code coverage in % for unit tests

Architecture and documentation

As with **C++**, Java is an object oriented language. You could use your **C++** skills to create this project.
It is important to take the time to define a simple architecture, that is progressive and without code duplication.

You are required to provide a clear and simple documentation of the the conception of your project.
You can also add a few conception diagrams: class diagram, sequence diagram,...., but there is no need to make class or sequence diagrams for the entire project. Instead, choose the important parts to be understood and which need to be documented.

Graphical Interface

You probably want to start creating a nice and friendly graphical interface.
In Java, it's not such a good idea. There aren't really any practical libraries and the result is often disappointing, given the necessary effort...

The objectives is not to make a graphical client; you can absolutely be satisfied with a console-mode client, with actions, via a readline.

Still, if you really want to make a graphical interface, you can do it as a bonus.

Bonus

There are several interesting bonuses to be done.
We have already looked at the graphical interface, but you can absolutely spend time on other bonuses, like the following:

- handling a champion with several games,
- creating a little AI in order to make the clients be handled by the computer (in this case you can check out the libraries to implement expert systems).
- more or less advanced rules
- Learner mode
- Automatic game advice
- Android app to play
- ...

In short, have fun :)