PARALLELISM AND CONCURRENCY
PRACTICAL ASSIGNMENT #3 WINTER TERM 2024
DISTRIBUTED PROGRAMMING: C/S and RMI

A. TRIVIAL SOLITAIRE C/S VERSION

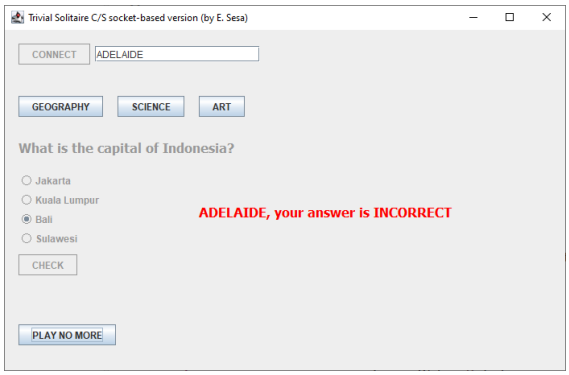
Design and write a client/server application to play "Trivial Solitaire".

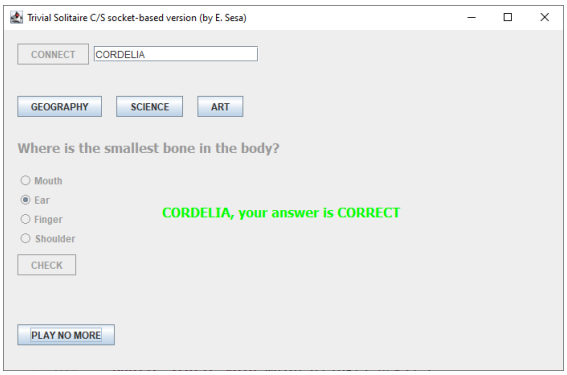
The server stores a number of questions of three different topics: ART, SCIENCE and GEO(graphy).

The client(s) gets in touch with the server (establishes a connection), sends a HELLO request and waits until the server answers back. Then, it sends NEXT GEO, NEXT SCIENCE or NEXT ART requests. To each of these requests, the server answers with a string representing a question of the requested type[1]. Questions sent to the client are chosen randomly and no question is sent twice. When all questions of a type have been sent to the client, the server replies with a NO MORE QUESTIONS OF THIS TYPE response. To finish the dialogue with the server, the client sends a STOP request. For each question the client shows its text, its four possible answers and checks the user's selection. If the server gets an unknown request (different from NEXT, HELLO or STOP) or a misplaced one (a NEXT before a HELLO or a HELLO after the first HELLO or a HELLO after a NEXT…) it sends a BAD REQUEST reply and closes the connection. The same happens when a NEXT request has an unknown type (different from GEO, SCIENCE or ART).

Your application must follow a "conversational continuity" approach so that several clients can be served simultaneously and without interference.

A non-GUI-based test client is already provided. Analyse and use it to test your server. Afterwards complete the GUI-based client.





[1] Class Question provides methods for building a string from a question (toString) and a question from a String (fromString). Also notice that a question instance not only includes the text of the question but also the four possible answers and an integer that specifies which of the four is the correct one.

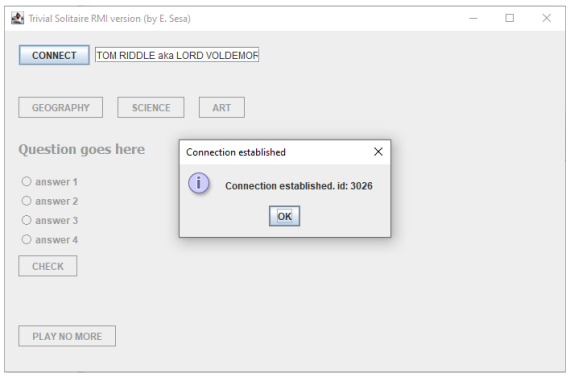
B. TRIVIAL SOLITAIRE RMI VERSION

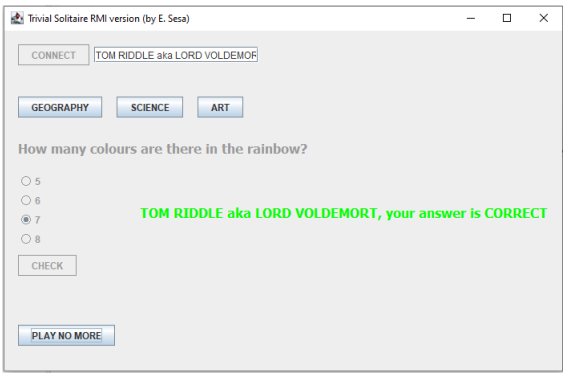
Write the same application using RMI. Complete the server implementation and the client.

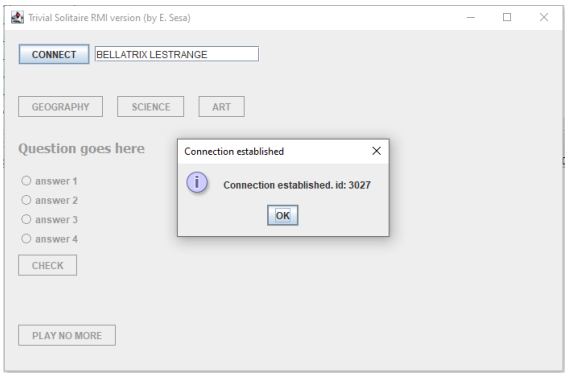
Remember that the server-side of the application must be able to play several games simultaneously, with different players (clients). That's why games have ids.

The following interface is suggested (you may change it but it is not really necessary)

```
public interface TrivialSolitaire extends Remote {

	public int Hello () throws RemoteException;
	// starts a new game. The result is the id of the newly created game
	// this id will have to be supplied in future interactions with the "server"

	public Question next (int id, String type) throws RemoteException;
	// request a question of the type of the second argument
	// retorn null if no question of that type is left

	public void stop(int id) throws RemoteException;
	// quits the game with the given id

}
```







SUBMISSION

Due date & how-to details: Check eCampus