Assignment 3: A Phrase Guessing Game with Client Failure Detection using Java RMI

Due Date: Monday April 3, before midnight
Demo: April 4-5, dates and details for registration for demo will be announced.
Weight: 10%

Description

For this assignment, you must augment your implementation of the RMI based client-server distributed application for a phrase guessing game to detect and handle client failures.

The functional requirements for the distributed application are identical to assignment 2. You will add failure detection and recovery mechanisms for the RMI client. First, you need implement a basic request deduplication mechanism. You must also make sure client records are removed if a client is suspected as crashed. The client must send the server heart-beats via a remote method invocation. The server must clean-up the client record in case it detects that the client has failed to send a heart-beat in a timely manner. Tutorial 6 provides an example for how client failure detection can be implemented over RMI. You can use it as starting point for this assignment.

Implementation Details

Similar to assignment 2, you must use Java RMI for communication between the client and server programs. You must not explicitly use socket programming for any part of your implementation.

Server The the game server must be implemented in a class that implements the `PhrasseGuessingGameServer` remote interface. This interface is similar to what you had for assignment 2, with the addition of the heartbeat method, and extra parameters to make method invocations idempotent. You can augment this interface with additional remote methods if they are required in your implementation.

The main entry point to the server program must first start the naming service (if not already started), and then create an instance of the phrase guessing game server. It must then registers the game server with the rmiregistry naming service.

After the game server is registered with the naming service, the server checks the list of client records and cleans up the records for the clients that have failed to send a heart-beat in a timely manner.

To make method invocations idempotent, there must be a sequence number added to the parameter list. When signing-up/logging-in the client must indicate a starting sequence number. The server must keep track of the last sequence number seen during a game, and it should not carry out the method operation if the sequence number is outdated. It should increase the expected next sequence number otherwise.

The following is one suggested approach. Feel free to explore alternative approaches. In an infinite loop, the server first sleeps for a given interval, then it goes through the list of client records and if the flag for the heartbeat is not set, it removes the record. It resets the flag for the clients that are still alive. It then goes to sleep again to repeat the process afterwards.

```
import java.rmi.*;

public interface PhraseGuessingGameServer extends Remote{
    public String startGame(
            String player,
            int number_of_words,
            int failed_attempt_factor
            int seq
    ) throws RemoteException;
    public String guessLetter(String player, int seq, char letter) throws RemoteException;
    public String guessPhrase(String player, int seq, String phrase) throws RemoteException;
    public String endGame(String player, int seq) throws RemoteException;
    public String restartGame(String player, int seq) throws RemoteException;
    public String addWord(int seq) throws RemoteException;
    public String removeWord(int seq) throws RemoteException;
    public String checkWord(int seq) throws RemoteException;

    public Boolean heartBeat(String player);
}
```

Client The client program must get the name and address of the phrase guessing game remote object as input argument, and get the reference to the remote object using the `Naming.lookup` method.

After obtaining the remote object reference and starting the game via a remote method invocation, the client must spawn a thread for sending heart-beats in regular intervals.

The main thread of the client program should continue with the game play by implementing the interaction between the user and the server via method calls to the remote phrase guessing game object.

To make method invocations idempotent, a sequence number must be included in the parameter list. When a client sends a request, it will use a sequence number, that is increased only when the remote method invocation has successfully returned. The starting point for the sequence number can be given to the server upon sign-up/sign-in for the player.

To show that your deduplication implementation works, you must include a mock remote method invocation duplicator in your client program. for each remote method invocation, your client program must repeat the method invocation with repeat sequence number with a 50% chance.

Documentation and Coding Standards

There is no official course standard for documentation and coding style. However, your code must be clearly documented, and written following a consistent formatting for indentation and naming styles. Your code must be self-documenting, well formatted, logically structured and easy to modify. The exact method by which each method works must be concisely but clearly and fully explained. Exactly how to do this is left to your discretion, but seek guidance from your instructor if you are unsure. Poorly formatted or documented code will loose mark on this component.

What to Hand In

Hand in a single packaged zip file with all your source code. Your package must include the source code for the remote interface(s), their implementation(s), the client class, the server class, the word repository class, and any other helper classes you have developed. You must also include your protocol design document that briefly outlines the components, the direction of method invocations, the data exchange (parameters are sent from the client to the server, and the return value is sent from the server to the client), and the actions that result from the invocations.

Hand in a single packaged zip file that includes the following:

1. The communication protocol design document for your distributed application. The design document must briefly outline the components, the direction of method invocations, the data exchange (parameters that are sent from the client to the server, and the return value that is sent from the server to the client), and the actions that result from the invocations. When you are asked to present your own design, you must explain the reasoning behind your design choices that shows critical thinking and understanding of goals and challenges in distributed systems.

2. Your program source code and a readme.txt file with instructions for running the program. You must include the source code for all the classes you have developed, including the client, the server, the word repository, the user accounts, and any other helper classes you have developed.

3. This assignment may be completed in groups. Each group must submit a statement of contribution that clearly states the contribution from each

team member. Submit your code package and your statement of contribution to the D2L dropbox for assignment 2.

Outcomes

Once you have completed this assignment:

- You can develop a distributed application in Java that uses Java RMI for inter-process communication;
- You know how to implement a stateful server and how to keep track of data and synchronize access to shared data;
- You can develop a distributed application with Java RMI that can detect client crashes.
- You can develop basic mechanism to deduplicate remote method invocations.

Rubric (100 marks)

1. Documentation 10
2. Clear description of communication protocol pertaining to failure detection 10
3. Client implementation, including invocation of heart-beats and random duplicate remote invocations 30
4. Server implementation 50
   (a) 10 implementation of the PhraseGuessingGameServer remote interface, instantiation and exporting servant(s)
   (b) 15 implementation of the failure detector for PhraseGuessingGameServer
   (c) 15 implementation of the idempotent RMI methods
   (d) 10 deleting the game state for crashed clients