Assignment 2: A Phrase Guessing Game with Remote Method Invocation in Java

Due Date: Wednesday, March 8, before midnight
Demo: Dates and details for registration for demos will be announced.
Weight: 10%

Description

For this assignment, you must once again develop a client-server distributed application in Java for a phrase guessing game. But this time, for the interaction between the client and the server, you must use remote method invocation on distributed objects instead of socket programming.

The functional requirements for the distributed application are identical to assignment 1. The client starts a game by specifying the game level they want to play. The server then chooses a number of words from a dictionary based on the game level, and presents the client with the structure of the phrase to be guessed. The client (the player) tries to guess the words chosen by the server by suggesting letters (one letter at a time) or the whole phrase. If the client suggests a letter that occurs on the phrase, the server places the letter in all its positions; otherwise the counter of allowed failed attempts is decremented. At any time the client is allowed to guess the whole phrase. A failed attempt occurs either when a suggested letter does not appear in the phrase, or when the suggested whole phrase does not match.

The client wins when the client completes the phrase, or guesses the whole phrase correctly. The server wins when the counter of allowed failed attempts reaches zero. The server should compute the total score of games using a score counter: if the client wins the score counter for that client is incremented, if the client loses the score counter for that client is decremented.

The server should keep track of client game history so returning clients are able to see their previous score, and new rounds of games update client score.

The server upon initialization builds a repository of words based on a standard Unix words file. The word repository must be implemented as a microservice that runs as a stand-alone java program. The server must allow the client to add a word to the repository, or remove a word from the repository, or check if the repository already contains a given word.

Implementation Details

You must use Java RMI for communication between the client, the server, and the word repository programs. You must not explicitly use socket programming for any part of your implementation.

Server The game server must be implemented in a class that implements the `PhrasseGuessingGameServer` remote interface. This interface must specify the following remote methods. 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 register the game server with the naming server.

```
import java.rmi.*;

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

Server Handling Multiple Players Simultaneously After you have made sure your server is able to handle a single client, extend it so that it can handle multiple players simultaneously. Multithreading is handled implicitly in RMI invocations. But you need to think carefully about how to keep track of the state of the game for different players. Look at the Bank Manager example in tutorial 6 for ideas. Make sure you synchronize methods that are accessing shared data on the server program.

Hint: Think how you can keep the state of the game for each player in a separate object in memory or on disk, and how you can get the reference to these objects based on player identification. For example, the game state object must have a field for the player id. A HashMap can be used to efficiently find the reference to the game state for each player.

Microservice for Word Repository

You must develop the word repository as a microservice similar to assignment 1. But for this assignment, the services must be offered via RMI. The word repository server must implement the following interface. The steps for creating an instance of the word repository object and registering it with the RMI registry are similar to the game server.

```
import java.rmi.*;

public interface WordRepositoryServer extends Remote{
        public boolean createWord(String word) throws RemoteException;
        public boolean removeWord(String word) throws RemoteException;
        public boolean checkWord(String word) throws RemoteException;
        public String getRandomWord(int length) throws RemoteException;
}
```

Microservice for User Accounts

Similar to assignment 1, you must implement a microservice for keeping track of the user accounts and game history. You must define the RMI interface and implement it as a separate class. The steps for creating an instance of the user account service and registering it with the RMI registry are similar to the game server. You can decide whether this service can be accessed by the client as well as the game-play server, or if it is only accessible via the game server. *Clearly document your design and the reasoning for your design choice in your protocol design document.*

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.

The client program should then implement the interaction between the user and the server via method calls to the remote phrase guessing game object. For this assignment, the client must not directly communicate with the word repository. Similar to assignment 1, a simple command line client user interface should allow the user to start a game specifying the level of difficulty, and during a game, input their choice of a letter or the whole phrase. The client must also give the option to the user to add, remove and check whether a word is in the repository. In each interaction during a game, the following should be displayed to the player:

- a current view of the phrase;
- the current value of the failed attempts counter;
- and the current value of the total score for this player.

Bonus: Scoreboard Microservice To further challenge yourself, implement a microservice that computes a scoreboard that ranks all players based on their

game history. You must define an RMI interface for the scoreboard service and implement it. The scoreboard interface must include at least one method that returns the current scoreboard for the top n players. The steps for creating an instance of the word repository object and registering it with the RMI registry are similar to the game server.

You can choose whether the services for the scoreboard can be directly accessed by the client, or the server program will act as the relay. Your protocol design document must clearly document your design and reasoning behind it.

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 purpose and services provided by each method 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 lose 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 define remote interfaces using java.rmi.Remote; how to implement the interface(s); how to develop a server that serves as a container for remote objects; you know to bind/lookup remote objects by name using the RMI naming service rmiregistry; you know how to run an RMI application;
- You know how to implement a stateful server and how to keep track of data and synchronize access to shared data;
- You can compare the experience of programming a client-server application, once using message passing with sockets, and once with remote method invocation.

Rubric (100 marks)

1. Documentation 10
2. Clear communication protocol design, and reasoning for design choices that shows critical thinking and understanding of goals and challenges in distributed systems 10
3. Client implementation 30
4. Server implementation 50
   (a) 15 implementation of the Phrase Guessing Game Server remote interface
   (b) 10 remote object instantiation and registration with RMI registry (exporting servant(s))
   (c) 5 keeping track of the state of the game for multiple players
   (d) 10 implementation of the Word Repository Server remote interface, instantiation and registration (exporting servant(s)) as a microservice
   (e) 10 implementation of the User Account Server remote interface, instantiation and registration (exporting servant(s)) as a microservice
   (f) 10 (Bonus) implementation of scoreboard microservice