Master of Software Systems Program

CICS 525 | Real-Time and Distributed Systems | May – July 2011

Assignment #2

General guidelines for homework: Before starting on this homework, review the homework guidelines provided on the first day of class. Remember that it is encouraged to discuss the problems with others in the class, but the assignment group only should complete all write-ups. **Homework grades will be based not only on getting the "correct answer," but also on good writing style and clear presentation of your solution**. It is your responsibility to make sure that the graders can easily follow your line of reasoning. Even if you can't solve the problem, you will receive partial credit for explaining why you got stuck on a promising line of attack. More importantly, you will get valuable feedback that will help you learn the material. Please acknowledge the people with whom you discussed the problems and what sources you used to help you solve the problem (e.g., books from the library).

Java RMI

The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language.

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application.

Distributed object applications need to do the following:

Locate remote objects. Applications can use various mechanisms to obtain references to remote objects. For example, an application can register its remote objects with RMI's simple naming facility, the RMI registry. Alternatively, an application can pass and return remote object references as part of other remote invocations.

Communicate with remote objects. Details of communication between remote objects are handled by RMI. To the programmer, remote communication looks similar to regular Java method invocations.

Load class definitions for objects that are passed around. Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's data.

A more complete tutorial for using Java RMI is available from Oracle/Sun <http://download.oracle.com/javase/tutorial/rmi/index.html>. You will use Java RMI to implement the stock trading system you built earlier.

A Client-Server Application - Simple Stock Trading System

You must implement a stock trading client-server application using Java RMI. The application involves two client types and one server.

Requirements

Here is a description of the functionality of your application:

1. The server keeps track of stock prices according to stock names. It returns stock prices when queried with the name of the stock.
2. One client type can update the price of individual stocks.
3. The other client type can obtain stock prices and buy or sell stocks.
4. The server keeps track of stock prices according to stock names. It returns stock prices when queried with the name of the stock. The server is assumed to be a non-terminating application.
5. The server also retains information about the stock owned by individual users and their cash balance.
6. At the server, stock prices are regularly updated for those stock tickers that clients have expressed an interest in. Initially the server is not tracking any stock. When the first client connects and requests a quote, the server begins to track that stock, and other stocks from then onward. The server retrieves stock prices from some online source (you can choose a method to achieve this). The prices are updated periodically every two minutes. For every new stock that the server tracks, it starts with 1000 shares to trade. Clients may purchase from these 1000 shares and no more.
7. The server makes all data persistent: when the server is terminated all relevant data is stored to disk and the server restores the data when restarted.
8. Each client is associated with a user. When a client connects to the server the first message from the client to the server is the username, which is indicated by the string USER <USERNAME>. If the user is new, then $1000 is credited to the user and the username is added to the server's data set.
9. A client can query stock prices by passing the stock name. The server returns the price of the stock if the ticker symbol is valid. A query is placed using a method like `query(ticker_name)`.
10. Clients can also buy or sell stocks with a method like `buy(ticker_name, num_stocks)` or `sell(ticker_name, num_stocks)`. The server should verify that the user has a sufficient balance for a buy transaction and that the user owns stock for the sell transaction. The transactions alter the user's cash balance appropriately.

The main requirement is to use Java RMI and invoke methods on the server using this infrastructure. Clients therefore will obtain a reference to the server object and then invoke the appropriate methods to update stock price and to buy/sell stocks.

Submission

You must submit your assignment using `handin`. The name of the course is `cics525` and the assignment name is `assignment2`. You must submit the following files (please use these names for your files):

1. All source code files;
2. A makefile with compilation instructions: Makefile;
3. A readme file in plain ASCII text format: Readme.txt;
4. The readme file must contain the following information:

   Lab number

   Names and student numbers

   Operating system

   Java compiler (version)

   Compilation instructions

   Execution instructions

   Known bugs