In-Class Exercise on Client Callback
October 7

Consider an RMI application where processes must be notified by an object server when a certain event occurs. For instance, in a chat room application, participants should be notified when a new user has joined the chat room. In a real-time financial trading system, traders must be notified when the prices reach certain values for the stocks in which they are interested. In the framework of the basic RMI API we have seen in class, it is not possible for the server to initiate a call to the client to transmit some information to the client when the information becomes available. This is because a remote method call is unidirectional, from client to the server. One way to implement this transmission of information from server to client is for each client to **poll** the object server by repeatedly invoking a remote method provided by the server, e.g., *hasStockPriceChangedAtLeastThisMuch*, until the method returns a non-zero value. But polling is a very expensive technique. Each remote method invocation uses non-trivial system resources and network resources. A more efficient technique is the **callback**. Every object client that is interested in the occurence of an event can register itself with the object server so that the server may initiate a remote method invocation to the object clients when the event of interest occurs.

In RMI, **client callback** allows an object client to register itself with a remote object server for **callbacks** so that the server can issue a remote method invocation to the client when certain events occur. With client callbacks, the remote method invocations are now two-way. Additional syntax is required to support this.

When an object server makes a callback, the roles are reversed: the object server becomes a client of the object client, in order to initiate a remote method invocation to the object client.

Your task:

Augment the *Hello* example provided so that each client object will register with the server for callback, and then will be notified later whenever another client object registers with the server for callback.

Steps:

(1) Client-side implementation for client callback

The client must provide a remote method that the server can invoke to notify it of the event. The client must provide a remote interface, e.g., *ClientInterface*, which should contain a method to be invoked by the server for the callbacks.

```
public String notify(String message) throws RemoteException;
```

This method *notify* is to be invoked by the server when it makes the callbacks, passing as argument a string. The client can take the received string and compose a new one to return to the server.

The client remote interface needs to be implemented.

In the object client class, add code to instantiate an object of the remote client interface implementation. A reference to the object is registered with the server using a remote method provided by the server, e.g., *registerForCallbacks*(*ClientInterface*).

The client unregisters itself after a user-specified interval of time, by invoking the remote method *unregisterForCallbacks*, to stop receiving callbacks from the server. This is useful

in applications where users leave the application, such as a chat room.

(2) Server-side implementation for client callback

A remote method is provided to allow a client to register for callback.
public void registerForCallback(ClientInterface clientObject)
A reference to an object that implements the client remote interface is accepted as an argument. Another method *unregisterForCallback* may also be provided to allow a client to terminate the callbacks.

The server needs to use a data structure, say *Vector* but you may use any data structure you wish, to maintain a list of the client interface references registered for callbacks. Each call to *registerForCallback* adds a reference to the vector, each call to *unregisterForCallback* removes a reference from the vector.

The server also implements a method *doCallbacks*(), which makes a callback to each client in the vector, reporting to the client the current number of registered clients. This method is called whenever a new client registers with the server. As you can imagine, the callbacks can be triggered by other events and may be made in an event handler.