Lab 3
Callbacks using RMI

In this lab you will be enhancing the shared whiteboard distributed application with the capability of the server informing the clients whenever an object is added. A naive way for clients to keep their view of the shared whiteboard up-to-date is for the client to continuously do *polling* of the server, by sending periodic messages to the server to ask for the current version number. A better and more efficient way to maintain the most recent version of the whiteboard is to use the mechanism of *callbacks*. You will implement a callback mechanism to add to the programs provided.

1 Callbacks

Instead of clients polling the server to find out whether an event has occurred, the server should inform the clients whenever an event occurs — this is the idea behind callbacks.

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.

2 Your task

Augment the shared whiteboard programs (as discussed in class) provided to you in the folder *rmi_whiteboard* so that each client will register with the server for callback, and then will be notified by the server whenever an graphical object has been added.

The *WhiteboardCallback* interface could be defined as:

```
public interface WhiteboardCallback implements Remote {
    void callback(int version) throws RemoteException;
}
```

This interface is implemented as a remote object by the client, enabling the server to send the client a version number whenever a new graphical object is added. But before the server can do this, the client needs to inform the server about its callback object. To make this possible, the *ShapeList* interface requires additional methods such as *register* and *unregister*, defined as:

```
int register(WhiteboardCallback callback) throws RemoteException;
int unregister(int callbackId) throws RemoteException;
```

After the client has obtained a reference to the remote object with the *ShapeList* interface and created an instance of its callback object, it uses the *register* method of *ShapeList* to inform the server that it is interested in receiving callbacks. The *register* method returns an integer (the *callbackId*) referring to the registration. When the client is finished it should call *unregister* to inform the server that it no longer requires callbacks. The server keeps a list of interested clients and notifying all of them each time its version number increases.

Show the TA: Run your programs to demonstrate that whenever a client adds a graphical object to the *ShapeList*, all the other clients get notified of this event via callbacks from the server.