Online tutorials:
Java RMI callback:
http://mkaroune.e-monsite.com/pages/rmi-1/rmi-callback.html
https://docs.oracle.com/cd/E13211_01/wle/rmi/callbak.htm

4: Event-driven programming using Server-side Callback: Java RMI Auction Server

Use Java's Remote Method Invocation to create an online auction.
People can join the auction, leave the auction, and bid for an item in the auction.
The current price of an item is updated as soon as anybody bids a valid price for the item.
You need to create an auction server. The server should support at least the following methods:

- join auction
- leave auction
- bid on an item.

You can read the initial list of items and minimum prices from a file or from standard input. When somebody joins the auction, you need to send the items on sale and their current price.
Then you need to create a Java client. The client joins the auction, displays the items for sale along with their current price, and lets the user place a bid for a particular item. Anyone should be able to join the auction with an arbitrary name – no authentication needed!
Since multiple clients will be bidding at the same time, the methods on the server must be properly synchronized. Whenever any client makes a successful bid, the current price at all the clients must reflect this change.
Implement your solution based on callback methods. Callback methods are used to inform a component about specific upcoming events, here the approach of a price update.
Test your application by **running client and server on different machines** with the **server** being **started** from a **local drive** (not a network drive)!

If you encounter problems with remote access using RMI:
Two issues may especially arise in the context of server-side callbacks:

- It may happen that the client tries to access an IP address different from the one on which the remote object was exported, see assignment 1 or http://www.javacodegeeks.com/2013/11/two-things-to-remember-when-using-java-rmi.html
  This is especially relevant for callbacks. Here, the client must also be assigned the system property java.rmi.server.hostname (set to the IP address of the client), otherwise the client might hand over (with the join() method) a wrong IP address to the server (e.g. that of a virtual host).
- If client and server are running in different networks: It may happen that the ports used by RMI are blocked when accessing a local network (e.g. the one the client is running on). It is not necessarily practical to release all ports (this may even not be possible). But the local network might be opened on specific ports, for example, port 80 is often open, e.g. to run a web server that would be accessible from the outside. If open ports to the network are

known, the remote object on the client could be deployed on this port (possibly as admin user if it is a privileged port). This can be done with the **`exportObject()`** method of **`UnicastRemoteObject`**:

`static Remote exportObject(Remote obj, int port)`

Exports the remote object to make it available to receive incoming calls, using the particular supplied port.