***Title      Homework 5: Automated Teller Machine***

***Author     Kalpan Bhatt***

***Date         01/08/2012***

***Overview***

**Purpose**

Develop the back end of a distributed Automated Teller Machine (ATM) system using basic Java networking.

**Brief description**

We are developing a distributed Automated Teller Machine (ATM). The ATM will be hosted in a different process than the client application. The client will connect to the server using Java networking and communicate with the server using a protocol.

**Functional Requirements**

The ATM will simulate a real world automated teller machine.

The ATM must support the following operations:

* deposit: add some dollar amount to account balance
* withdrawal: deduct some dollar amount from account balance
* balance inquiry: get current account balance

The ATM will run in its own process and will handle remote requests from a client over a socket connection running in some other process.

**Design**

ATM interface will be implemented by both the real ATM and a client side proxy for the ATM, called a stub. The client will use a stub that will proxy all the operations of the ATM and delegate them an implementation object on the server.

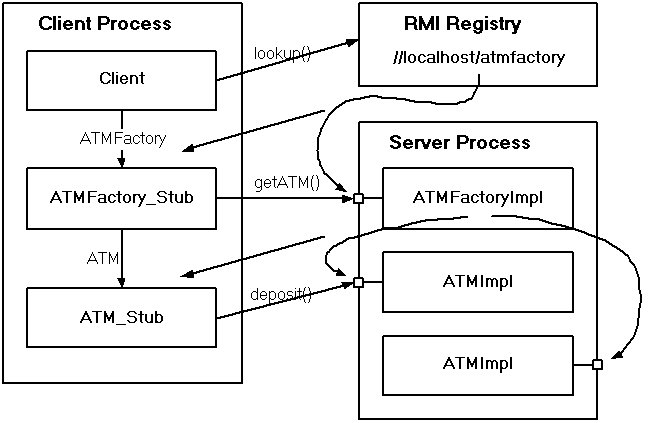
The ATM will now include several individual Accounts and each of the ATM methods must now include a parameter that allows the account number to be specified. For simplicity we'll assume the account identifier type is int. In this system, we will create an ATM factory that the client will use to get a reference to a remote ATM.

The ATM factory is a server side object with a remote interface just like the ATM. We will create an ATMFactory interface that has a single method getATM() that returns a remote reference to the serverside ATM instance.

The server process will start up, create an ATMFactoryImpl instance, and then register it with the RMI naming service or registry using java.rmi.Naming. The client will then be able to lookup the ATM factory and connect to it. Then the client will use the getATM() method of the factory to get a remote reference to an ATM instance.

**Architecture**

The following diagram shows how the server will process incoming client requests.



**Goals**

1. All classes should be in package cscie160.hw5
2. Enhance ATM interface to be an RMI remote interface.
3. Create an RMI remote implementation ATMImpl that implements the remote ATM interface.
4. Create an Account class. It will hold information about its balance.
5. Create an ATMFactory interface that is RMI ready and supports a single getATM() method.
6. Create the RMI ready implementation of the factory interface. It should return a remote reference to an ATM instance.
7. Create a server and Client class.
8. Client class should have main method for the demonstration.

**See Also**

[**http://courses.dce.harvard.edu/~cscie160/hw5-05.htm**](http://courses.dce.harvard.edu/~cscie160/hw5-05.htm)

**Assumptions**

1. *For simplicity we'll assume the account identifier type is int.*

**Current procedure/functionality**

1. Server process register ATMFactory as remote object in RMIRegistry.
2. When a client will be started on the same host and will get access to the ATM implementation and remote method call of ATM via remote object of ATMImplementation.
3. ATMFactoryImpl has implementation of public interface ATMFactory. This will give instance of remote object ATM.
4. ATMImplementation will have methods to get information about Account balance.
5. There are currently 3 methods implemented to Deposit, Withdraw and QueryBalance.

**How to run the assignment**

*Unzip the contents of the jar file HW5.jar*

*Run the Ant tool at the folder level of build.xml*

*To demonstrate your system you will need to start three distinct processes.*

*First startup the rmi registry:*

*C:\> rmiregistry*

*Then startup the server:*

*C:\> java cscie160.hw5.Server*

*Once the server is up and running, start up the client:*

*C:\> java cscie160.hw5.Client*

*You should see the output as mentioned in the output.txt file*