***Title      Homework 6: Automated Teller Machine***

***Author     Kalpan Bhatt***

***Date         01/01/2013***

***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

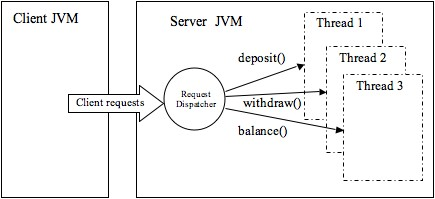
The ATM will run in its own process and will handle remote requests from a client over a socket connection running in some other process.

**Design**

In the ATM system, the client and ATM will be running in different processes. Since the client cannot reference memory in the server process, it cannot get an actual reference to the real ATM object running on the server. Instead the client will use a proxy that presents all the behavior that characterizes an ATM. In fact, the client doesn't even need to know if the actual ATM is local or remote. This is achieved using a Java interface.

**Architecture**

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



**Goals**

1. All classes should be in package cscie160.hw6
2. These classes remains the same as per HW4: ATM, ATMProxy, ATMException, ATMImplementation, Client
3. Create a class ATMRunnable that is responsible for executing client requests. This will parse the request to determine the type of operation to execute and then invoke ATM interface.
4. Modify Account class to handle synchronization.
5. Update server class. It should create a pool of threads to handle incoming requests.
6. Client class should have main method for the demonstration.

**See Also**

[**http://courses.dce.harvard.edu/~cscie160/hw6-06.htm**](http://courses.dce.harvard.edu/~cscie160/hw6-06.htm)

**Assumptions**

1. *Currently the system supports only one Account.*
2. *Assume you should have 5 Threads in this pool.*

**Current procedure/functionality**

1. Server process will listen for connections from clients on port 1099.
2. When a client will be started on the same host and sends a request on port 1099, the request will be received by ATMProxy class.
3. ATMProxy has implementation of public interface ATM. The same interface is also implemented by server side ATMImplementation class.
4. ATMProxy will call methods of the ATMImplementation.
5. ATMImplementation will have methods to get information about Account balance.
6. There are currently 3 methods implemented to Deposit, Withdraw and QueryBalance.

**New Functionality:**

1. ATMThreads will service the remote requests. When the Threads are initially started, they will be waiting for notification that an element of work (an ATMRunnable) requires processing.
2. The ServerSocket is created and accepts an incoming unit of work.
3. Once Server has read an incoming request, it will package up that request into an ATMRunnable.
4. Once the ATMRunnable has been created, it will be added to the LinkedList collection. It will be this data structure that ATMThreads will be waiting on.
5. After adding your ATMRunnable to the list, it will notify waiting threads that a request must be processed. The act of notifying waiting Threads will cause an ATMThread to resume execution.
6. After notifying waiting threads, server will resume reading additional incoming requests from the client.

**How to run the assignment**

*Unzip the contents of the jar file HW6.jar*

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

*Use the jar file created in build folder to run server and client files.*