Ryan Mitchell
CSCIE160 Project
Jan 22nd, 2013

To Run Project, open four terminals and execute the commands, from /source, in order:

- rmiregistry
- java cscie160.Project.BankServer
- java cscie160.Project.ATMServer
- java cscie160.Project.Client

There are 14 classes created for the Project (excluding new exception types and enum Commands). They can be categorized in three ways:

Things that are sent over RMI as objects (and implement serializable):
AccountInfo
Account
TransactionNotification

Things that communicate with each other (Extend parent interfaces, implement unicastRemoteObject):
Client (extends ATMListener)
BankImpl (extends Bank)
SecurityImpl (extends Security)
ATMImpl (extends ATM)

Servers:
BankServer
ATMServer
Client

Notes

In homework 6, the Account object had methods for withdrawing and depositing amounts. However, these methods were removed in the project -- because the new account balance must be explicitly updated through the Bank (Account is only passed by value by the remote object Bank, over RMI) it made sense to perform calculations in the ATM, and submit only the final account balance to Bank, along with an account number, and use the Account.setBalance method.

A small modification was made to the Client. Rather than passing an ATM object between all the methods, a global ATM object is used. This might be changed if a client could register itself with multiple ATMs, but this scenario does not make physical sense.

Because Integers are not designed to preserve leading zeros, they make non-ideal holders for account numbers. I found that the leading zeros in Integers would often be preserved within the same object, but once you start passing Integer Account numbers back and forth through RMI, the leading zeros are dropped.
I decided to use a String instead. Although it’s an unusual choice of variable type for something that is a “number,” because we are performing no mathematical operations on the account numbers there is no reason to use an int, float, Integer, or other similar type. I did not change PINs to Strings, and this may cause problems for PINs with leading zeros in the future.

In the Security object, authorizeDeposit, authorizeWithdraw, and authorizeBalance all take account numbers as arguments, rather than AccountInfo objects. At the point that these are called, the pin has already been verified, and only the account number is needed.
Similarly, the Bank does not receive AccountInfo objects or Account pins because it also does not handle pin authentication.

Unlike the other interfaces created (which contain ALL of the methods that their child classes contain), ATMListener only has one method “receiveTransactionNotification.” To add all of Client’s methods to ATMListener would make it far too specific -- there are many things that are hard-coded into Client that do not necessarily represent a “listener” (for instance: testATM()). I chose to make ATMListener contain only the method that represents the actual listening element -- receiving the transactions. While Client implements ATMListener, it contains much more functionality.