CS 535 OO Programming & Design
Fall Semester, 2013
Assignment 5

San Diego State University -- This page last updated 10/31/13

Assignment 5
Due Nov 16, 11:59 pm

1. (25 points - 15 working/meets requirements, 5 style/design, 5 unit tests) Banks do millions if not billions of transactions a day. Any round-off in calculations can amount to a lot of money. So all calculations done on a bank account balance have to be exact. To ensure that this is the case create a Currency class. The bank account balances are to be currency objects, not numbers. The currency class will do addition and subtraction without floating point round off. You need to be able to create a currency object from a string and a number. You need to be able to add/subtract currency objects and add/subtract numbers to/from currency objects. Currency objects need to be comparable. That is they must implement the methods in the Magnitude class. Once created a currency object should not be able to change its value. Currency objects need to print themselves out as dollars (see below). An example of some of the functionality of the currency class can be seen in the following test case.

```
| a b c d sum |
a := Currency fromString: '1.00'.
b := Currency fromNumber: 2.12.
sum := Currency fromNumber: 2.12.
c := a + b.
self assert: sum = c.
d := c - 1.00.
self assert: d = b.
self assert: a printString = '$1.00'.
```

You might find it convenient to implement convenience methods so the above test can be rewritten as:

```
| a b c d sum |
a := '1.00' asCurrency.
b := 2.12 asCurrency.
sum := 2.12 asCurrency.
c := a + b.
self assert: sum = c.
d := c - 1.00.
self assert: d = b.
self assert: a printString = '$1.00'.
```

2. (15 points - 5 working/meets requirements, 5 style/design, 5 unit tests) Our bank has two types of customers, normal and preferred. Preferred customers get overdraft protection. When normal customers try to withdraw more money than is in their account the transaction is cancelled and the customer is charged a $5.00 fee. The fee could make the account balance go negative. Preferred customers are allowed withdraw $1,000.00 more than their account holds without penalty. Any transaction that makes the balance of a preferred customers balance become less than -$1,000.00 is cancelled and the customer is charged a $3.00 fee.

3. (45 points - 25 working/meets requirements, 15 style/design, 5 unit tests) Money is a serious business for banks. They need to keep record of all transactions so they can show were money came from and where it went. When you deposit a check into your account the money is put on hold until the the check clears. Until the check clears your account has two balances. First is the total balance. That is the sum of all income minus the sum of all expenditures. Second is the available balance. That is the total amount minus all funds that are on hold. Depending on their origin checks take differing amounts of time to clear. Sometimes checks bounce. That is the account they are written on does not have enough money in the account when the check reaches the bank. When this happens the account is deducted a fee (see problem 2) and the check is revoked. Revoking a check means that the amount of the check is deducted from the account it was deposited into.

   Add/modify the following instance methods to your BankAccount class. For testing purposes the names of the following methods must be as they are given below. The name of your class must be BankAccount. See the end of the assignment description for information about Durations.

   balance
   Returns the current total balance in the account now

   balanceIn: aDuration
   Returns the total balance in the account in aDuration amount of time from now. aDuration can be negative.

   availableBalance
   Returns the current available balance in the account now

   availableBalanceIn aDuration
   Returns the current available balance in the account in aDuration amount of time from now. aDuration can be negative.

   transactionsFrom: aFilename
   Processes the transactions in the given file on the current account. That is reads the file and then changes the balance of the account based on the transactions in the file.

   Your BankAccount class needs the following **class** method. For testing purposes the name of the following method must be as given below.

   fromFile: aFilename
   Reads the transaction data from the given file and returns BankAccount created using that data. Filename is a filename object for a file with transactions in it.

The input file with transactions contains four types of transactions: NewAccount, Deposit, Withdrawal, Cancel. Each transaction is on a single line. Each item of a transaction is separated by a tab. All transactions start with the following:

transactionID transactionTimestamp transactionType

Where

transactionID is an integer. Transaction IDs are unique.

transactionTimestamp is a time stamp giving the time and date of the transaction. The format of a timestamp is

mm.dd.yyyy hh:mm

For example 10.15.2013 17:30 is the timestamp for October 15, 2013 5:30:00 PM. The year and date are separated by a space. See below for information about timestamps.

transactionType is one of NewAccount, Deposit, Withdrawal, Cancel

There can be only one NewAccount transaction in the transaction file, but there does not have to be one in the file. If there is one in the file it has to be the first transaction. The format of NewAccount transaction is:

transactionID transactionTimestamp NewAccount name customerType amount

Where "name" can have multiple parts, each separated by spaces. "customerType" is either "Normal" or "Preferred". "amount" is a dollar amount like: "$25.00". A sample NewAccount transaction is:

1253 10.15.2013 17:30 NewAccount Roger Whitney Normal $250.13

The format of the Cancel transaction is:

transactionID transactionTimestamp Cancel transactionIDToCancel

Where "transactionIDToCancel" is the transaction id of the transaction to cancel. This is primarily used for checks that have bounced. The transaction that is canceled can be either a deposit or a withdrawal. A sample Cancel transaction is:

182126 10.15.2013 17:30 Cancel 17433

The format of the Withdrawal transaction is:

transactionID transactionTimestamp Withdrawal amount

Where "amount" is a dollar amount like: "$25.00". A sample Withdrawal transaction is:

1534 10.17.2013 19:31 Withdrawal $13.15

The format of the Deposit transaction starts with:

transactionID transactionTimestamp Deposit type

"type" is either "Cash" or "Check". If type is "Check" then the full transaction is:

transactionID transactionTimestamp Deposit Check holdDurationInDays amount

Where "amount" is a dollar amount like: "$25.00". "holdDurationInDays" is an integer indicating how many days the deposit will be on hold. A sample "Deposit Check" transaction is:

7362 10.30.2013 19:31 Deposit Check 3 $13.15

If type is "Cash" then the full transaction is:

transactionID transactionTimestamp Deposit Cash amount

Where "amount" is a dollar amount like: "$25.00". A sample "Deposit Cash" transaction is:

7362 10.30.2013 19:31 Deposit Cash $13.15

A sample transaction file may look like:

```
1253 10.15.2013 17:30 NewAccount Roger Whitney Normal $250.13
1334 10.17.2013 15:31 Deposit Check 4 $13.15
2001 10.21.2013 13:59 Deposit Check 3 $13.15
1982 10.20.2013 19:31 Deposit Cash $13.15
2010 10.21.2013 03:30 Cancel 1334
4103 10.29.2013 19:29 Withdrawal $13.15
```

Note that the transactions in the file are not necessarily in order by transaction ID or by timestamp.

Durations and Timestamps

You will find Timestamps and Durations useful in this assignment. Duration represents an interval of time - days, hours, minutes, seconds. One can creates a duration by sending the following messages to numbers: days, hours, minutes, seconds. Here are some examples:

```
3 days
1.3 days
1.5 hours
```

Timestamps represent a date and time, for example October 16, 2013 2:00:06 PM. There are methods in the Timestamp class. You can compare Timestamp objects with <, >, =, >=, <=. You can also add durations to Timestamps to get another Timestamp object. For example:

```
Timestamp now +  3 days  + 12 minutes
```

One can also create Timestamps from strings. We will use the format month.date.year hour:minute. For example:

```
Timestamp readFromDateAndTime: ('10.15.2013 17:30' readStream)
```

Will give us a Timestamp object for October 15, 2013 5:30:00 PM.

How to turn in the assignment

In your image create Package called Assignment5. Make sure that all the code for this assignment is in your Assignment5 package. When you add a method to existing Smalltalk classes only include in your package the methods that you write. You will upload your Assignment5 package to your store account.