CIS 279 HW4 Inheritance, Polymorphism and Abstract Classes

In this project, you'll have an opportunity to create an abstract base class from which two other classes are derived. You will need to override an abstract method of the base class and its toString() method. As an added bonus you'll create a LinkedList of Transaction objects. Please create the following:

- An abstract base class named Account which declares all common attributes and defines all common methods for its derived classes
- A concrete class named CheckingAccount derived from Account
- A concrete class named Mortgage derived from Account
- A class named Transaction that records checks and deposits for checking accounts and payments for mortgages
- A LinkedList of Transaction objects in the base class and a method for iterating through this list and invoking the toString() method of the Transaction class
- A driver class that:
    - creates two objects of CheckingAccount and two objects of Mortgage
    - adds them to a LinkedList of the Account class
    - creates four transaction objects for each of the CheckingAccount and Mortgage objects and adds them to their transaction lists
    - iterates through the LinkedList using an extended for statement and invokes the toString() method of each object

Processing

The Account class defines most of the tasks which involve processing transactions and calculating interest. The CheckingAccount class adds no data members; it needs nothing more than the Account class provides. It must override the abstract processTransaction method of the base class. This method will only add deposits to the CheckingAccount balance, subtract checks from it and add the transaction object to a LinkedList of Transactions.

The processing required by the Mortgage class makes it more complicated. Mortgages have periodic payments and use a somewhat complicated formula to calculate them. The payments themselves are a bit complex because a portion of a payment reflects interest on the outstanding balance of the loan and the periodic payment minus this interest reflects the amount repaid. The relative amounts of interest and prepayment change each month as the balance declines. Therefore, this class needs a method to calculate the current month's interest which is the balance times the monthly interest rate. Then it must calculate the amount of load repaid by subtracting the current month's interest from the periodic payment. After calculating the amount repaid, use it to reduce the balance. These three operations must be performed in the correct order. Here's how I coordinated them:

```
public void processTransaction( Transaction transactionObject)
{
        transactionList.addLast( transactionObject); // Add a transaction to the list.

        // Calculate interest the amount repaid and change the account balance.
```

```
        if ( transactionObject.getTransactionType() == PAYMENT)
        {
                setCurrentMonthInterest( balance * monthlyInterestRate );

                setBalanceRepaid(periodicPayment - currentMonthInterest) ;

                setBalance( balance - balanceRepaid );

        }

        // To simplify this, we avoided dealing with an invalid transaction type.
    }
```

The following method should be an old friend:

```
    public void calcPeriodicPayment ()
    {
        double annuityFactor =
            (( 1 - ( 1 / Math.pow((1 + monthlyInterestRate ), termInMonths)))
            / monthlyInterestRate);

        periodicPayment = balance / annuityFactor;
    }
```

These UML diagrams should help you:

| abstract class Account |
|---|
| # customerID : int // # indicates protected which derived classes can access directly |
| # accountNumber : int |
| # accountType : char // 'c' for checking, 'm' for mortgage |
| # interestRate : double // probably 0 for checking accounts |
| # monthlyInterestRate : double // interestRate / 12 |
| # currentMonthInterest : double |
| # balance : double |
| # LinkedList<Transaction> transactionList // List of Transaction objects |
| + abstract processTransaction( transactionObject : Transaction) |
| + setCustomerID(customerID : int) |
| + getCustomerID() : int |
| + setAccountNumber(accountNumber : int) |
| + getAccountNumber() : int |
| + setAccountType(accountType : char) // 'c' indicates checking; 'm' indicates mortgage |
| + getAccountType() : char |
| + setInterestRate( interestRate : double ) |
| + getInterestRate() : double |
| + setCurrentMonthInterest(currentMonthInterest : double) |
| + getCurrentMonthInterest() : double |
| + setBalance(balance : double) // to be overridden by derived classes |
| + getBalance () : double |

| |
|---|
| + listTransactions() : String // transactionObject.toString() |
| + toString() : String // Displays the values of the base class data members |

| derived class CheckingAccount extends Account |
|---|
| |
| // No other attributes needed |
| + processTransaction( transactionObject : Transaction) // overrides base class method |
| // No need to override base class toString() : String |

| derived class Mortgage extends Account |
|---|
| - termInYears : int |
| - termInMonths : int |
| - periodicPayment : double |
| - balanceRepaid : double |
| + processTransaction( transactionObject : Transaction) // overrides base class method |
| + setTerm(term : int) // sets both termInYears and termInMonths |
| + getTerm() : int |
| + calcPeriodicPayment () : void |
| + setCurrentMonthInterest(currentMonthInterest : double) |
| + getCurrentMonthInterest() : double |
| + setBalanceRepaid( balanceRepaid : double ) |
| + getBalanceRepaid (): double |
| + toString() : String // overrides base class toString(). It invokes the base class method<br>// to display its data members then displays the values of the<br>// members it has added. |

| class Transaction |
|---|
| - transactionID : int // can be check number for a check or just an ID number for<br>// checking account deposits and mortgage payments. |
| - transactionDate : Date |
| - transactionAmount: double |
| - transactionType: char // 'D' – deposit: increases as checking balances<br>// 'C' – check: reduces checking balances<br>// 'P' – payment: reduces mortgage balances |
| + setTransactionID( transactionID : int ) |
| + getTranasctionID() : int |
| + setDate(transactionDate : Date) |
| + getDate() : Date |
| + setTransactionAmount(amount : double) |
| + getTransactionAmount() : double |
| + setTransactionType( transactionType : char) |
| + getTransactionType( ) : char |