import java.util.*;

/**
 * A Bank object simulates the behavior of a simple bank/ATM.
 *
 * The visit method opens this Bank for business, prompting the customer
 * for input. The visit method opens the next Bank account, in case of
 * an error.
 *
 * @see BankAccount
 *
 * @version 5
 */

public class Bank
{
    private String bankName;           // the name for this Bank
    private Terminal atm;              // for talking with the customer
    private int balance = 0;           // total cash on hand
    private int transactionCount = 0;  // number of Bank transactions
    private Month month;               // the current month.

    private TreeMap accountList;      // mapping names to accounts.

    // what the banker can ask of the bank

    private static final String BANKER_COMMANDS =
            "Banker commands: exit, open, customer, report, help."
            + "Private static final String CUSTOMER_TRANSACTIONS =
            "    Customer transactions: deposit, withdraw, transfer, balance, cash check, quit, help."
            + "    Customer transactions: 
            else if (command.startsWith( "c" ) ) {
                BankAccount acct = whichAccount();
                if ( acct != null )
                    processTransactionsForAccount( acct );
            } else {
                // Unrecognized Request
                atm.println( "unknown command: " + command );
            }
            report();
            atm.println( "Goodbye from " + bankName );
        }
        // Construct a Bank with the given name and Terminal.
        public Bank( String bankName, Terminal atm )
        {
            this.atm = atm;
            this.bankName = bankName;
            accountList = new TreeMap();
            month = new Month();
        }

        /*
        * Open a new bank account, prompting the user for information.
        *
        * @param bankName the name for this Bank.
        * @param atm this Bank's Terminal.
        */
        public Bank( String bankName, Terminal atm )
        {
            this.atm = atm;
            this.bankName = bankName;
            accountList = new TreeMap();
            month = new Month();
        }

        /*
        * Construct a Bank with the given name and Terminal.
        */
        private void openNewAccount()
        { [snip]...
```java
break;
case 'f':
    newAccount = new FeeAccount(startup, this);
    break;
case 'r':
    newAccount = new RegularAccount(startup, this);
    break;
default:
    atm.println("invalid account type: " + accountType);
    return;
}
accountList.put(accountName, newAccount);
atm.println("opened new account " + accountName + " with $" + startup);

// Prompt the customer for transaction to process. Then send an appropriate message to the account.
private void processTransactionsForAccount(BankAccount acct) {
    help(CUSTOMER_TRANSACTIONS);

    String transaction;
    while (!transaction.equals("quit")) {
        if (transaction.startsWith("h")) {
            help(CUSTOMER_TRANSACTIONS);
        } else if (transaction.startsWith("d")) {
            int amount = atm.readInt("    amount:");
            atm.println("    deposited " + acct.deposit(amount));
        } else if (transaction.startsWith("w")) {
            int amount = atm.readInt("    amount:");
            atm.println("    withdrew " + acct.withdraw(amount));
        } else if (transaction.startsWith("c")) {
            int amount = atm.readInt("    amount of check: ");
            atm.println("    cashed check for " + ((CheckingAccount)acct).honorCheck(amount));
        } else if (transaction.startsWith("t")) {
            atm.print("    to ");
            BankAccount toacct = whichAccount();
            if (toacct != null) {
                int amount = atm.readInt("    amount to transfer: ");
                atm.println("    transfered " + toacct.deposit(acct.withdraw(amount)));
            }
        } else if (transaction.startsWith("b")) {
            atm.println("    current balance " + acct.requestBalance());
        } else {
            atm.println("    sorry, unknown transaction" );
        }
    }
    atm.println();
}

// Prompt for an account name (or number), look it up in the account list. If it's there, return it; otherwise report an error and return null.
private BankAccount whichAccount() {
    String accountName = atm.readWord("account name: ");
    BankAccount account = (BankAccount) accountList.get(accountName);
    if (account == null) {
        atm.println("not a valid account");
    }
    return account;
}

// Action to take when a new month starts. Update the month field by sending a next message. Loop on all accounts, sending each a newMonth message.
private void newMonth() {
    month.next();
    // for each account
    //    account.newMonth()
}

// Report bank activity. For each BankAccount, print the customer id (name or number), account balance and the number of transactions. Then print Bank totals.
private void report() {
    atm.println(bankName + " report for " + month);
    atm.println(”Summaries of individual accounts:" );
    atm.println(”account  balance   transaction count" );
    for (Iterator i = accountList.keySet().iterator(); i.hasNext(); ) {
        String accountName = (String) i.next();
        BankAccount acct = (BankAccount) accountList.get(accountName);
        atm.println(accountName + ”	" + acct.getBalance() + ”	” + acct.getTransactionCount());
    }
    atm.println(”Bank totals”);
    atm.println(”open accounts: " + getNumberOfAccounts());
    atm.println(”cash on hand: $" + getBalance());
    atm.println(”transactions:  " + getTransactionCount());
    atm.println();
}
```
Welcome the user to the bank and instruct her on her options.

```java
private void instructUser()
{
atm.println( "Welcome to " + bankName );
atm.println( "Open some accounts and work with them." );
help( BANKER_COMMANDS );
}
```

`help` function displays a help string.

```java
private void help( String helpString )
{
atm.println( helpString );
atm.println();
}
```

Increment bank balance by given amount.

```java
public void incrementBalance(int amount)
{
balance += amount;
}
```

Increment by one the count of transactions for this bank.

```java
public void countTransaction()
{
transactionCount++;}
```

Get the number of transactions performed by this bank.

```java
public int getTransactionCount()
{
return transactionCount ;
}
```

Get the current bank balance.

```java
public int getBalance()
{
return balance;
}
```

Get the current number of open accounts.

```java
public int getNumberOfAccounts()
{
return accountList.size();
}
```

Run the simulation by creating and then visiting a new `Bank`.

```java
public static void main( String[] args )
{
// parse the command line arguments for the echo flag and the name of the bank

boolean echo    = false;         // default does not echo
String bankName = "Faithless Trust"; // default bank name

for (int i = 0; i < args.length; i++ ) {
    if (args[i].equals("-e")) {
        echo = true;
    } else {
        bankName = args[i];
    }
}

Bank aBank = new Bank( bankName, new Terminal(echo) );
aBank.visit();
```
public abstract class BankAccount

private int balance = 0; // Account balance (whole dollars)
private int transactionCount = 0; // Number of transactions performed.
private Bank issuingBank; // Bank issuing this account

public BankAccount( int initialBalance, Bank issuingBank )
{
    this.issuingBank = issuingBank;
deposit( initialBalance );
}

public int withdraw( int amount )
{
    incrementBalance( -amount );
countTransaction();
return amount ;
}

public int deposit(int amount)
{
incrementBalance( amount);;
countTransaction();
return amount ;
}

public int requestBalance()
{
countTransaction();
return getBalance() ;
}

public int getBalance()
{
return balance;
}

public void incrementBalance( int amount )
{
balance += amount;
this.getIssuingBank().incrementBalance( amount );
}

public int getTransactionCount()
{
return transactionCount;
}

/**
* Construct a BankAccount with the given initial balance and issuing Bank. Construction counts as this BankAccount's first transaction.
*
* @param initialBalance the opening balance.
* @param issuingBank the bank that issued this account.
*/

public BankAccount( int initialBalance, Bank issuingBank )
{
    this.issuingBank = issuingBank;
deposit( initialBalance );
}
public int getTransactionCount()
{
    return transactionCount;
}

/**
* Increment by 1 the count of transactions, for this account
* and for the issuing Bank.
*/

public void countTransaction()
{
    transactionCount++;
    this.getIssuingBank().countTransaction();
}

/**
* Get the bank that issued this account.
* Does NOT count as a transaction.
* 
* @return issuing bank.
*/

public Bank getIssuingBank()
{
    return issuingBank;
}

/**
* Action to take when a new month starts.
*/

public abstract void newMonth();
Copyright 2003 Bill Campbell and Ethan Bolker

A RegularAccount does nothing when the next month starts.

Action to take when a new month starts.

*super* InitialTransaction, transaction.

super RegularTransaction( InitialTransaction, bank Transaction).

**/}

public class RegularTransaction extends Transaction

/**
   * In do nothing //
   */

public void endTransaction() {

    /*
     * A RegularTransaction is a BankTransaction that has no special behavior.
     */

    super endTransaction.

}

}
/* Copyright 2003 Bill Campbell and Ethan Bolker
 * 
 * A CheckingAccount is a BankAccount with one new feature: 
 * the ability to cash a check by calling the honorCheck method. 
 * Each honored check costs the customer a checkFee. 
 * 
 * @version 5
 */

public class CheckingAccount extends BankAccount
{
    private static int checkFee = 2;  // pretty steep for each check

    /**
     * Constructs a CheckingAccount with the given
     * initial balance and issuing Bank.
     * Counts as this account's first transaction.
     *
     * @param initialBalance the opening balance for this account.
     * @param issuingBank the bank that issued this account.
     */
    public CheckingAccount( int initialBalance, Bank issuingBank )
    {
        super( initialBalance, issuingBank );
    }

    /**
     * Honor a check:
     * Charge the account the appropriate fee
     * and withdraw the amount.
     *
     * @param amount amount (in whole dollars) to be withdrawn.
     * @return the amount withdrawn.
     */
    public int honorCheck( int amount )
    {
        incrementBalance( - checkFee );
        return withdraw( amount );
    }

    /**
     * Action to take when a new month starts.
     */
    public void newMonth()
    {
    }
}

/**
 * Copyright 2003 Bill Campbell and Ethan Bolker
 * */
public void newMonth() 

/** 
 * A FeeAccount is a BankAccount with one new feature: 
 * the user is charged for each transaction. 
 */

public class FeeAccount extends BankAccount

private static int transactionFee = 1;

/** 
* Constructor, accepting an initial balance and issuing Bank. 
* @param initialBalance the opening balance. 
* @param issuingBank the bank that issued this account. 
*/

public FeeAccount( int initialBalance, Bank issuingBank )

super( initialBalance, issuingBank);

/** 
* The way a transaction is counted for a FeeAccount is different: 
* it levies a transaction fee as well as counting the transaction. 
*/

public void countTransaction()

incrementBalance( - transactionFee );

super.countTransaction();

/** 
* Action to take when a new month starts. 
*/

public void newMonth()


/* joi/5/bank/class Month */

import java.io.*;
import java.util.Calendar;

/**
 * The Month class implements an object that keeps track of the month of the year.
 *
 * @version 5
 */

public class Month {

private static final String[] monthName =

private int month;
private int year;

/**
 *  Month constructor constructs a Month object initialized to the current month and year.
 */

public Month() {
    Calendar rightNow = Calendar.getInstance();
    month = rightNow.get( Calendar.MONTH );
    year  = rightNow.get( Calendar.YEAR );
}

/**
 *  Advance to next month.
 */

public void next() {
    // needs completion
}

/**
 *  How a Month is displayed as a String - for example, "Jan, 2003".
 *
 * @return String representation of the month.
 */

public String toString() {
    //
    }

/**
 * For unit testing.
 */

public static void main( String[] args ) {
    Month m = new Month();
    for (int i=0; i < 14; i++, m.next()) {
        System.out.println(m);
    }
    for (int i=0; i < 35; i++, m.next()); // no loop body
    System.out.println("three years later: " + m);
    for (int i=0; i < 120; i++, m.next()); // no loop body
    System.out.println("ten years later: " + m);
}
}