import java.util.*;

/**
 * A Bank object simulates the behavior of a simple bank/ATM.
 * It contains a Terminal object and a collection of
 * BankAccount objects.
 *
 * The visit method opens this Bank for business,
 * prompting the customer for input.
 *
 * To create a Bank and open it for business issue the command
 * <code>java Bank</code>.
 *
 * @see BankAccount
 * @version 5
 */

public class Bank
{
private String bankName;           // the name of 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.

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.");

/**
 * Construct a Bank with the given name and Terminal.
 *
 * @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();
}

/**
 * Simulates interaction with a Bank.
 * Presents the user with an interactive loop, prompting for
 * banker transactions and in case of the banker transaction
 * "customer", an account id and further customer
 * transactions.
 */

public void visit()
{
instructUser();

String command;
while (!(command = atm.readWord("banker command: "))).equals("exit") ) {

if (command.startsWith("h")) {
help( BANKER_COMMANDS );
}
else if (command.startsWith("o")) {
openNewAccount();
}
else if (command.startsWith( "r") ) {
BankAccount acct = whichAccount();
if ( acct != null )
processTransactionsForAccount( acct );
}
else {
// Unrecognized Request
atm.println("unknown command: " + command);
}
}
report();
atm.println("Goodbye from " + bankName);
}

// Open a new bank account,
// prompting the user for information.

private void openNewAccount()
{
String accountName = atm.readWord("Account name: ");
char accountType = atm.readChar("Checking/Fee/Regular? (c/f/r): ");
int startup = atm.readInt("Initial deposit: ");
BankAccount newAccount;
switch( accountType ) {
case 'c':
newAccount = new CheckingAccount( startup, this );
break;
case 'f':
newAccount = new FeeAccount( startup, this );
break;
case 'r':
newAccount = new RegularAccount( startup, this );
break;
default:
throw new IllegalArgumentException("Unknown account type: " + accountType);
}
accountList.put( accountName, newAccount );
}

private BankAccount whichAccount()
{
if ( atm.readWord("Account name: ") == null ) return null;
return (BankAccount) accountList.get( atm.readWord("Account name: "));
}

private void processTransactionsForAccount( BankAccount acct )
{
if ( acct.getBalance() == 0 )
atm.println("Account has no money.");
else
atm.println("Transaction result: ");
acct.accept( atm );
}

private void instructUser()
{
atm.printLine("Welcome, open a new bank account,
 // or talk to a banker.
 // For help, try: help.");
}

private void report()
{
report( "Customer: ");
report( "Banker: ");
}

private void report( String prefix )
{
int i;
for (i = 0; i < prefix.length(); i++)
atm.printChar( prefix.charAt(i) );
for (BankAccount acct : accountList.values())
acct.report();
}
}
113break;
114case 'f':
115newAccount = new FeeAccount( startup, this );
116break;
117case 'r':
118newAccount = new RegularAccount( startup, this );
119break;
120default:
121atm.println( "invalid account type: " + accountType);
122return;
123}
124accountList.put( accountName, newAccount );
125atm.println( "opened new account " + accountName
126+ " with $" + startup );
127}
128
129// Prompt the customer for transaction to process. 
130// Then send an appropriate message to the account.
131private void processTransactionsForAccount( BankAccount acct )
132{
133help( CUSTOMER_TRANSACTIONS );
134
135String transaction;
136while (!(transaction =
137atm.readWord("    transaction: ")).equals("quit")) {
138
139if ( transaction.startsWith( "h" ) ) {
140help( CUSTOMER_TRANSACTIONS );
141}
142else if ( transaction.startsWith( "d" ) ) {
143int amount = atm.readInt( "    amount: ");
144atm.println("    deposited " + acct.deposit( amount ));
145}
146else if ( transaction.startsWith( "w" ) ) {
147int amount = atm.readInt( "    amount: ");
148atm.println("    withdrew " + acct.withdraw( amount ));
149}
150else if ( transaction.startsWith( "c" ) ) {
151int amount = atm.readInt( "    amount of check: ");
152atm.println("    cashed check for " +
153((CheckingAccount)acct).honorCheck( amount ));
154}
155else if ( transaction.startsWith( "t" ) ) {
156atm.print("    to ");
157BankAccount toacct = whichAccount();
158if (toacct != null) {
159int amount = atm.readInt("    amount to transfer: ");
160atm.println("    transfered " +
161toacct.deposit(acct.withdraw(amount)));
162}
163}
164else if ( transaction.startsWith( "b" ) ) {
165atm.println("    current balance " +
166acct.requestBalance());
167}
168else {
169atm.println("    sorry, unknown transaction" );
170}
171atm.println();
172}
173// Prompt for an account name (or number), look it up
174// in the account list. If it's there, return it;
175// otherwise report an error and return null.
176private BankAccount whichAccount()
177{
178String accountName = atm.readWord( "account name: ");
179BankAccount account = (BankAccount) accountList.get(accountName);
180if (account == null) {
181atm.println("not a valid account");
182}
183return account;
184}
185
186// Action to take when a new month starts.
187// Update the month field by sending a next message.
188// Loop on all accounts, sending each a newMonth message.
189private void newMonth()
190{ 
191month.next();
192// for each account
193//      account.newMonth()
194}
196// For each account, print the customer id (name or number),
197// account balance and the number of transactions.
198// Then print Bank totals.
199private void report()
200{ 
201atm.println( bankName + " report for " + month );
202atm.println( "Summaries of individual accounts: ");
203atm.println( "account  balance  number of transactions" );
204for (Iterator i = accountList.keySet().iterator();
205i.hasNext(); ) {
206String accountName = (String) i.next();
207BankAccount acct = (BankAccount) accountList.get(accountName);
208atm.println(accountName + "  $" + acct.getBalance() + "  	" +
209acct.getTransactionCount());
210}
211atm.println( "Bank totals");
212atm.println( "open accounts: " + getNumberOfAccounts() );
213atm.println( "cash on hand: $" + getBalance());
214atm.println( "transactions:  " + getTransactionCount());
215}
private void instructUser()
{
atm.println( "Welcome to " + bankName );
atm.println( "Open some accounts and work with them." );
help( BANKER_COMMANDS );
}

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

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

public void countTransaction()
{
transactionCount++;  
}

public int getTransactionCount()
{
return transactionCount;
}

public int getBalance()
{
return balance;
}

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

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();
// Copyright 2003 Bill Campbell and Ethan Bolker

/**
 * A BankAccount object has private fields to keep track of its current balance, the number of transactions performed and the Bank in which it is an account, and public methods to access those fields appropriately.
 *
 * @see Bank
 * @version 5
 */

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

/**
 * 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 );
}

/**
 * Withdraw the given amount, decreasing this BankAccount's balance and the issuing Bank's balance. Counts as a transaction.
 *
 * @param amount the amount to be withdrawn
 * @return amount withdrawn
 */

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

/**
 * Deposit the given amount, increasing this BankAccount's balance and the issuing Bank's balance. Counts as a transaction.
 *
 * @param amount the amount to be deposited
 * @return amount deposited
 */

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

/**
 * Request for balance. Counts as a transaction.
 *
 * @return current account balance.
 */

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

/**
 * Get the current balance.
 * Does NOT count as a transaction.
 *
 * @return current account balance
 */

public int getBalance()
{
return balance;
}

/**
 * Increment account balance by given amount. Also increment issuing Bank's balance.
 * Does NOT count as a transaction.
 *
 * @param amount the amount of the increment.
 */

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

/**
 * Get the number of transactions performed by this account. Does NOT count as a transaction.
 *
 * @return number of transactions performed.
 */

public int getTransactionCount()
{
return transactionCount;
}

/**
 * Request for transaction count. Counts as a transaction.
 *
 * @return current transaction count
 */

public int getRequestTransactionCount()
{
countTransaction();
return 0;
}

/**
 * Get the number of transactions performed and the Bank that issued this account.
 *
 * @return a BankAccount
 */

public BankAccount getIssuingBank()
{
return issuingBank;
}

/**
 * Deposit dollar amount into the BankAccount.
 *
 * @param amount amount to deposit
 */

private void deposit( int amount )
{
this.issuingBank.deposit( amount );
}

/**
 * Withdraw dollar amount from the BankAccount.
 *
 * @param amount amount to withdraw
 */

private void withdraw( int amount )
{
this.issuingBank.withdraw( amount );
}

/**
 * Get the number of transactions performed and the Bank that issued this account.
 *
 * @param amount amount to deposit
 */

private int depositAndWithdraw( int amount )
{
this.issuingBank.deposit( amount );
this.issuingBank.withdraw( amount );
return amount ;
}

/**
 * Make a deposit and withdraw the same amount.
 * Does NOT count as a transaction.
 *
 * @param amount amount to deposit and withdraw
 */

private int depositAndWithdraw(int amount)
{
this.issuingBank.deposit( amount );
this.issuingBank.withdraw( amount );
return amount ;
}

/**
 * Test methods to access those fields appropriately.
 */

public static void main( String[] args )
{
BankAccount bankAccount = new BankAccount();
bankAccount.deposit( 500 );
bankAccount.withdraw( 300 );
}
public int getTransactionCount()
{
    return transactionCount;
}

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

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

/**
 * Get the bank that issued this account.
 *
 * @return issuing bank.
 */

public Bank getIssuingBank()
{
    return issuingBank;
}

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

public abstract void newMonth();
public class RegularAccount extends BankAccount
{

   /**
    * 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 RegularAccount( int initialBalance, Bank issuingBank )
   {
      super( initialBalance, issuingBank );
   }

   /**
    * Action to take when a new month starts.
    *
    * A RegularAccount does nothing when the next month starts.
    */
   public void newMonth() {
      // do nothing
   }

   public RegularAccount( int initialBalance, Bank issuin...
public class CheckingAccount extends BankAccount
{
    private static int checkFee = 2; // pretty steep for each check

    public CheckingAccount( int initialBalance, Bank issuingBank )
    {
        super( initialBalance, issuingBank );
    }

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

    public void newMonth()
    {
    }
}

/**
 * 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:
 */
public int honorCheck( int amount )
{
    incrementBalance( -checkFee );
    return withdraw( amount );
}

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

/**
 * 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

    public CheckingAccount( int initialBalance, Bank issuingBank )
    {
        super( initialBalance, issuingBank );
    }

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

    public void newMonth()
    {
    }
}

/**
 * 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:
 */
public int honorCheck( int amount )
{
    incrementBalance( -checkFee );
    return withdraw( amount );
}

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

/**
 * 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 FeeAccount extends BankAccount
{
    private static int transactionFee = 1;

    public FeeAccount( int initialBalance, Bank issuingBank )
    {
        super( initialBalance, issuingBank);
    }

    public void countTransaction()
    {
        incrementBalance( - transactionFee );
        super.countTransaction();
    }

    public FeeAccount( int initialBalance, Bank issuingBank, String accountNumber )
    {
        super( initialBalance, issuingBank, accountNumber);
    }

    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()
// {
// }

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);
}
}