// BankAccount.java

/**
 * Copyright 2003 Bill Campbell and Ethan Bolker
 */

import java.io.Serializable;

/**
 * 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 9
 */

public abstract class BankAccount implements Serializable {
    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.
     *
     * @exception InsufficientFundsException when appropriate.
     */

    protected BankAccount(int initialBalance, Bank issuingBank) throws InsufficientFundsException {
        this.issuingBank = issuingBank;
        deposit(initialBalance);
    }

    /**
     * Get transaction fee. By default, 0.
     *
     * @return the fee.
     */

    protected int getTransactionFee() {
        return 0;
    }

    /**
     * The bank that issued this account.
     *
     * @return the Bank.
     */

    protected Bank getIssuingBank() {
        return issuingBank;
    }

    /**
     * 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     * @param amount the amount to be withdrawn
     * @return amount withdrawn
     *
     * @exception InsufficientFundsException when appropriate.
     */

    public int withdraw(int amount) throws InsufficientFundsException {
        incrementBalance(-amount - getTransactionFee());
        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
     *
     * @exception InsufficientFundsException when appropriate.
     */

    public int deposit(int amount) throws InsufficientFundsException {
        incrementBalance(amount - getTransactionFee());
        countTransaction();
        return amount;
    }

    /**
     * Request for balance. Counts as a transaction.
     *
     * @return current account balance.
     *
     * @exception InsufficientFundsException when appropriate.
     */

    public int requestBalance() {
        return balance;
    }

    /**
     * Processed the getBalance() request.
     * @param arg the unknown argument.
     * @return the result.
     */

    protected void processGetBalance(int arg) {
        // Do whatever.
    }

    /**
     * Processed the getIssuingBank() request.
     * @param arg the unknown argument.
     * @return the result.
     */

    protected void processGetIssuingBank(int arg) {
        // Do whatever.
    }

    /**
     * Processed the getTransactionCount() request.
     * @param arg the unknown argument.
     * @return the result.
     */

    protected void processGetTransactionCount(int arg) {
        // Do whatever.
    }

    /**
     * Processed the deposit(amount) request.
     * @param arg the unknown argument.
     */

    protected void processDeposit(int arg) {
        // Do whatever.
    }

    /**
     * Processed the withdraw(amount) request.
     * @param arg the unknown argument.
     */

    protected void processWithdraw(int arg) {
        // Do whatever.
    }

    // More stuff...
113} throws InsufficientFundsException 
114{" 
115incrementBalance( -getTransactionFee() ); 
116countTransaction(); 
117return getBalance() ; 
118} 
119
120/** 
121* Get the current balance. 
122* Does NOT count as a transaction. 
123* 
124* @return current account balance 
125*/ 
126
127public int getBalance() 
128{" return balance; 
129} 
130
132/** 
133* Increment account balance by given amount. 
134* Also increment issuing Bank’s balance. 
135* Does NOT count as a transaction. 
136* 
137* @param amount the amount of the increment. 
138* 
139* @exception InsufficientFundsException when appropriate. 
140*/ 
141
142public final void incrementBalance( int amount ) 
143throws InsufficientFundsException 
144{" int newBalance = balance + amount; 
145if (newBalance < 0) {
146throw new InsufficientFundsException(
147"for this transaction" ); 
148} 
149balance = newBalance; 
150getIssuingBank().incrementBalance( amount ); 
151} 
152
154/** 
155* Get the number of transactions performed by this 
156* account. Does NOT count as a transaction. 
157* 
158* @return number of transactions performed. 
159*/ 
160
161public int getTransactionCount() 
162{" return transactionCount; 
163} 
164
166/** 
167* Increment by 1 the count of transactions, for this account 
168* and for the issuing bank. 
169* Does NOT count as a transaction. 
170* 
171* @exception InsufficientFundsException when appropriate. 
172*/ 
173
174public void countTransaction() 
175throws InsufficientFundsException 
176{" transactionCount++; 
177this.getIssuingBank().countTransaction(); 
178} 
179
182/** 
183* Action to take when a new month starts. 
184* 
185* @exception InsufficientFundsException thrown when funds 
186* on hand are not enough to cover the fees. 
187* Action to take when a new month starts. 
188* 
189* @param account Fees on account when funds 
190* 
191* @exception InsufficientFundsException when appropriate. 
192*/ 
193
194public abstract void newMonth() 
195throws InsufficientFundsException; 
196}