

```

1 // fo1/9/bank/BankAccount.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.io.Serializable;
7
8 /**
9  * A BankAccount object has private fields to keep track
10 * of its current balance, the number of transactions
11 * performed and the Bank in which it is an account, and
12 * and public methods to access those fields appropriately.
13  *
14  * @see Bank
15  * @version 9
16  */
17
18 public abstract class BankAccount
19     implements Serializable
20 {
21     private int balance = 0; // Account balance (whole dollars)
22     private int transactionCount = 0; // Number of transactions performed
23     private Bank issuingBank; // Bank issuing this account
24
25     /**
26      * Construct a BankAccount with the given initial balance and
27      * issuing Bank. Construction counts as this BankAccount's
28      * first transaction.
29      *
30      * @param initialBalance the opening balance.
31      * @param issuingBank the bank that issued this account.
32      *
33      * @exception InsufficientFundsException when appropriate.
34      */
35     protected BankAccount( int initialBalance, Bank issuingBank )
36     {
37         throws InsufficientFundsException
38     {
39         this.issuingBank = issuingBank;
40         deposit( initialBalance );
41     }
42
43     /**
44      * Get transaction fee. By default, 0.
45      * Override this for accounts having transaction fees.
46      *
47      * @return the fee.
48      */
49     protected int getTransactionFee()
50     {
51         return 0;
52     }
53 }
54
55 /**
56  * The bank that issued this account.

```

```

57  *
58  * @return the Bank.
59  */
60
61     protected Bank getIssuingBank()
62     {
63         return issuingBank;
64     }
65
66     /**
67      * Withdraw the given amount, decreasing this BankAccount's
68      * balance and the issuing Bank's balance.
69      * Counts as a transaction.
70      *
71      * @param amount the amount to be withdrawn
72      * @return amount withdrawn
73      *
74      * @exception InsufficientFundsException when appropriate.
75      */
76
77     public int withdraw( int amount )
78     {
79         throws InsufficientFundsException
80     {
81         incrementBalance( -amount - getTransactionFee() );
82         countTransaction();
83         return amount ;
84     }
85
86     /**
87      * Deposit the given amount, increasing this BankAccount's
88      * balance and the issuing Bank's balance.
89      * Counts as a transaction.
90      *
91      * @param amount the amount to be deposited
92      * @return amount deposited
93      *
94      * @exception InsufficientFundsException when appropriate.
95      */
96     public int deposit( int amount )
97     {
98         throws InsufficientFundsException
99     {
100         incrementBalance( amount - getTransactionFee() );
101         countTransaction();
102         return amount ;
103     }
104
105     /**
106      * Request for balance. Counts as a transaction.
107      *
108      * @return current account balance.
109      *
110      * @exception InsufficientFundsException when appropriate.
111      */
112     public int requestBalance()

```

```

113     throws InsufficientFundsException
114     {
115         incrementBalance( - getTransactionFee() );
116         countTransaction();
117         return getBalance() ;
118     }
119 }
120 /**
121  * Get the current balance.
122  * Does NOT count as a transaction.
123  */
124  * @return current account balance
125  */
126
127     public int getBalance()
128     {
129         return balance;
130     }
131 }
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
142     public final void incrementBalance( int amount )
143     throws InsufficientFundsException
144     {
145         int newBalance = balance + amount;
146         if (newBalance < 0) {
147             throw new InsufficientFundsException(
148                 "For this transaction" );
149         }
150         balance = newBalance;
151         getIssuingBank().incrementBalance( amount );
152     }
153 }
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
161     public int getTransactionCount()
162     {
163         return transactionCount;
164     }
165 }
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
174     public void countTransaction()
175     throws InsufficientFundsException
176     {
177         transactionCount++;
178         this.getIssuingBank().countTransaction();
179     }
180 }
181 /**
182  * Action to take when a new month starts.
183  */
184  * @exception InsufficientFundsException thrown when funds
185  * on hand are not enough to cover the fees.
186  */
187
188     public abstract void newMonth()
189     throws InsufficientFundsException;
190 }

```