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



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68       * @param amount the amount to be withdrawn
69       * @return amount withdrawn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

```
113          countTransaction();
114          return getBalance() ;
115      }
116
117      /**
118       * Get the current balance.
119       * Does NOT count as a transaction.
120       *
121       * @return current account balance
122       */
123
124      public int getBalance()
125      {
126          return balance;
127      }
128
129      /**
130       * Increment account balance by given amount.
131       * Also increment issuing Bank's balance.
132       * Does NOT count as a transaction.
133       *
134       * @param amount the amount of the increment.
135       *
136       * @exception InsufficientFundsException when appropriate.
137       */
138
139      public final void incrementBalance( int amount )
140          throws InsufficientFundsException
141      {
142          int newBalance = balance + amount;
143          if (newBalance < 0) {
144              throw new InsufficientFundsException(
145                          "for this transaction" );
146          }
147          balance = newBalance;
148          getIssuingBank().incrementBalance( amount );
149      }
150
151      /**
152       * Get the number of transactions performed by this
153       * account. Does NOT count as a transaction.
154       *
155       * @return number of transactions performed.
156       */
157
158      public int getTransactionCount()
159      {
160          return transactionCount;
161      }
162
163      /**
164       * Increment by 1 the count of transactions, for this account
165       * and for the issuing Bank.
166       * Does NOT count as a transaction.
167       *
168       * @exception InsufficientFundsException when appropriate.
```



```
169       */
170
171      public void countTransaction()
172          throws InsufficientFundsException
173      {
174          transactionCount++;
175          this.getIssuingBank().countTransaction();
176      }
177
178      /**
179       * Action to take when a new month starts.
180       *
181       * @exception InsufficientFundsException thrown when funds
182       *            on hand are not enough to cover the fees.
183       */
184
185      public abstract void newMonth()
186          throws InsufficientFundsException;
187  }
```