```
// joi/9/bank/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
 * 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 performe
    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.
     * Override this for accounts having transaction fees.
     *
     * @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     * @param amount the amount to be withdrawn
     * @return amount withdrawn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()
```

```
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  }
```