Listing 7.2 BankAccount.java

/**
 * 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 7
 */

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.
 *
 * @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 * @param amount the amount to be withdrawn
 * @return amount withdrawn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()
throws InsufficientFundsException
{
incrementBalance( - getTransactionFee() );
return balance;
}
113 countTransaction();
114 return getBalance();
115 }

116 /**
117 * Get the current balance.
118 * Does NOT count as a transaction.
119 *
120 * @return current account balance
121 */
122
123 public int getBalance()
124 {
125 return balance;
126 }

127 /**
128 * Increment account balance by given amount.
129 * Also increment issuing Bank's balance.
130 * Does NOT count as a transaction.
131 *
132 * @param amount the amount of the increment.
133 *
134 * @exception InsufficientFundsException when appropriate.
135 */
136
137 public final void incrementBalance( int amount )
138 throws InsufficientFundsException
139 {
140 int newBalance = balance + amount;
141 if (newBalance < 0) {
142 throw new InsufficientFundsException(
143 "for this transaction" );
144 }
145 balance = newBalance;
146 getIssuingBank().incrementBalance( amount );
147 }

148 /**
149 * Get the number of transactions performed by this
150 * account. Does NOT count as a transaction.
151 *
152 * @return number of transactions performed.
153 */
154
155 public int getTransactionCount()
156 {
157 return transactionCount;
158 }

159 /**
160 * Increment by 1 the count of transactions performed by this
161 * account, and for the issuing bank.
162 * Does NOT count as a transaction.
163 *
164 * @exception InsufficientFundsException when appropriate.
165 */
166
167 public void countTransaction()
168 throws InsufficientFundsException
169 {
170 transactionCount++;
171 this.getIssuingBank().countTransaction();
172 }

173 /**
174 * Action to take when a new month starts.
175 *
176 * @exception InsufficientFundsException thrown when funds
177 * on hand are not enough to cover the fees.
178 * Does NOT count as a transaction.
179 *
180 * @param amount the amount of the increment.
181 *
182 * @exception InsufficientFundsException when appropriate.
183 */
184
185 public abstract void newMonth()
186 throws InsufficientFundsException;