```
 1  // joi/4/bank/Bank.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // Lines marked "///" flag places where changes will be needed.
 7
 8  /// import java.util.??
 9
10  /**
11   * A Bank object simulates the behavior of a simple bank/ATM.
12   * It contains a Terminal object and a collection of
13   * BankAccount objects.
14   *
15   * Its public method visit opens this Bank for business,
16   * prompting the customer for input.
17   *
18   * To create a Bank and open it for business issue the command
19   * <code>java Bank</code>.
20   *
21   * @see BankAccount
22   * @version 4
23   */
24
25  public class Bank
26  {
27      private String bankName;           // the name of this Bank
28      private Terminal atm;              // for talking with the customer
29      private int balance = 0;           // total cash on hand
30      private int transactionCount = 0;  // number of Bank transactions don
31
32      private BankAccount[] accountList; /// collection of BankAccounts
33      /// omit next line when accountList is dynamic
34      private final static int NUM_ACCOUNTS = 3;
35
36      // what the banker can ask of the bank
37
38      private static final String BANKER_COMMANDS =
39      "Banker commands: " +
40          "exit, open, customer, report, help.";
41
42      // what the customer can ask of the bank
43
44      private static final String CUSTOMER_TRANSACTIONS =
45      "    Customer transactions: " +
46          "deposit, withdraw, transfer, balance, quit, help.";
47
48      /**
49       * Construct a Bank with the given name and Terminal.
50       *
51       * @param bankName the name for this Bank.
52       * @param atm  this Bank's Terminal.
53       */
54
55      public Bank( String bankName, Terminal atm )
56      {
```



```
57          this.atm      = atm;
58          this.bankName = bankName;
59          // initialize collection:
60          accountList   = new BankAccount[NUM_ACCOUNTS]; ///
61
62          /// When accountList is an array, fill it here.
63          /// When it's an ArrayList or a TreeMap, delete these lines.
64          /// Bank starts with no accounts, banker creates them with
65          /// the openNewAccount method.
66          accountList[0] = new BankAccount(  0, this);
67          accountList[1] = new BankAccount(100, this);
68          accountList[2] = new BankAccount(200, this);
69      }
70
71      /**
72       * Simulates interaction with a Bank.
73       * Presents the user with an interactive loop, prompting for
74       * banker transactions and in case of the banker transaction
75       * "customer", an account id and further customer
76       * transactions.
77       */
78
79      public void visit()
80      {
81          instructUser();
82
83          String command;
84          while (!(command =
85                   atm.readWord("banker command: ")).equals("exit")) {
86
87              if (command.startsWith("h")) {
88                  help( BANKER_COMMANDS );
89              }
90              else if (command.startsWith("o")) {
91                  openNewAccount();
92              }
93              else if (command.startsWith("r")) {
94                  report();
95              }
96              else if (command.startsWith( "c" ) ) {
97                  BankAccount acct = whichAccount();
98                  if ( acct != null )
99                      processTransactionsForAccount( acct );
100             }
101             else {
102                 // Unrecognized Request
103                 atm.println( "unknown command: " + command );
104             }
105         }
106         report();
107         atm.println( "Goodbye from " + bankName );
108     }
109
110
111     // Open a new bank account,
112     // prompting the user for information.
```

```
113
114      private void openNewAccount()
115      {
116          /// when accountList is a dynamic collection
117          /// remove the next two lines, uncomment and complete
118          /// the code between /* and */
119          atm.println(bankName + " is accepting no new customers\n");
120          return;
121
122          /*
123          // prompt for initial deposit
124          int startup = atm.readInt( "Initial deposit: " );
125
126          // create newAccount
127          BankAccount newAccount = new BankAccount( startup, this );
128
129          //  and add it to accountList
130          ???
131
132          // inform user
133          atm.println( "opened new account " + ??? /// name or number
134                               + " with $" + newAccount.getBalance());
135          */
136      }
137
138      // Prompt the customer for transaction to process.
139      // Then send an appropriate message to the account.
140
141      private void processTransactionsForAccount( BankAccount acct )
142      {
143          help( CUSTOMER_TRANSACTIONS );
144
145          String transaction;
146          while (!(transaction =
147                   atm.readWord("    transaction: ")).equals("quit")) {
148
149              if ( transaction.startsWith( "h" ) ) {
150                  help( CUSTOMER_TRANSACTIONS );
151              }
152              else if ( transaction.startsWith( "d" ) ) {
153                  int amount = atm.readInt( "    amount: " );
154                  atm.println("    deposited " + acct.deposit( amount ));
155              }
156              else if ( transaction.startsWith( "w" ) ) {
157                  int amount = atm.readInt( "    amount: " );
158                  atm.println("    withdrew " + acct.withdraw( amount ));
159              }
160              else if (transaction.startsWith("t")) {
161                  atm.print( "    to ");
162                  BankAccount toacct = whichAccount();
163                  if (toacct != null) {
164                      int amount = atm.readInt("    amount to transfer: ");
165                      atm.println("    transfered " +
166                                  toacct.deposit(acct.withdraw(amount)));
167                  }
168              }
```



```
169              else if (transaction.startsWith("b")) {
170                  atm.println("    current balance " +
171                              acct.requestBalance());
172              }
173              else {
174                  atm.println("    sorry, unknown transaction" );
175              }
176          }
177          atm.println();
178      }
179
180      // Prompt for an account name (or number), look it up
181      // in the account list. If it's there, return it;
182      // otherwise report an error and return null.
183
184      private BankAccount whichAccount()
185      {
186          /// prompt for account name or account number
187          /// (whichever is appropriate)
188          int accountNumber = atm.readInt("account number: ");
189
190          /// look up account in accountList
191          /// if it's there, return it
192          /// else the following two lines should execute
193          if ( accountNumber >= 0 && accountNumber < NUM_ACCOUNTS ) {
194              return accountList[accountNumber];
195          }
196          else {
197              atm.println("not a valid account");
198              return null;
199          }
200      }
201
202      // Report bank activity.
203      // For each BankAccount, print the customer id (name or number),
204      // account balance and the number of transactions.
205      // Then print Bank totals.
206
207      private void report()
208      {
209          atm.println( "\nSummaries of individual accounts:" );
210          atm.println( "account  balance   transaction count" );
211          for (int i = 0; i < NUM_ACCOUNTS; i++ ) {              ///
212              atm.println(i + "\t" + accountList[i].getBalance() +     ///
213                          "\t" + accountList[i].getTransactionCount());///
214          }
215
216          atm.println( "\nBank totals");
217          atm.println( "open accounts: " + getNumberOfAccounts() );
218          atm.println( "cash on hand: $" + getBalance());
219          atm.println( "transactions:  " + getTransactionCount());
220          atm.println();
221      }
222
223
224      // Welcome the user to the bank and instruct her on
```

```
225      //  her options.
226
227      private void instructUser()
228      {
229          atm.println( "Welcome to " + bankName );
230          atm.println( "Open some accounts and work with them." );
231          help( BANKER_COMMANDS );
232      }
233
234      // Display a help string.
235
236      private void help( String helpString )
237      {
238          atm.println( helpString );
239          atm.println();
240      }
241
242      /**
243       * Increment bank balance by given amount.
244       *
245       * @param amount the amount increment.
246       */
247
248      public void incrementBalance(int amount)
249      {
250          balance += amount;
251      }
252
253      /**
254       * Increment by one the count of transactions,
255       * for this bank.
256       */
257
258      public void countTransaction()
259      {
260          transactionCount++;
261      }
262
263      /**
264       * Get the number of transactions performed by this bank.
265       *
266       * @return number of transactions performed.
267       */
268
269      public int getTransactionCount( )
270      {
271          return transactionCount ;
272      }
273
274      /**
275       * Get the current bank balance.
276       *
277       * @return current bank balance.
278       */
279
280      public int getBalance()
```



```
281      {
282          return balance;
283      }
284
285      /**
286       * Get the current number of open accounts.
287       *
288       * @return number of open accounts.
289       */
290
291      public int getNumberOfAccounts()
292      {
293          return NUM_ACCOUNTS; /// needs changing ...
294      }
295
296      /**
297       * Run the simulation by creating and then visiting a new Bank.
298       * <p>
299       * A -e argument causes the input to be echoed.
300       * This can be useful for executing the program against
301       * a test script, e.g.,
302       * <pre>
303       *   java Bank -e < Bank.in
304       * </pre>
305       *
306       * @param args the command line arguments:
307       *         <pre>
308       *         -e echo input.
309       *         bankName any other command line argument.
310       *         </pre>
311       */
312
313      public static void main( String[] args )
314      {
315          // parse the command line arguments for the echo
316          // flag and the name of the bank
317
318          boolean echo    = false;        // default does not echo
319          String bankName = "River Bank"; // default bank name
320
321          for (int i = 0; i < args.length; i++ ) {
322              if (args[i].equals("-e")) {
323                  echo = true;
324              }
325              else {
326                  bankName = args[i];
327              }
328          }
329          Bank aBank = new Bank( bankName, new Terminal(echo) );
330          aBank.visit();
331      }
332  }
```