```
// joi/7/bank/Bank.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.util.*;

/**
 * A Bank object simulates the behavior of a simple bank/ATM.
 * It contains a Terminal object and a collection of
 * BankAccount objects.
 *
 * The visit method opens this Bank for business,
 * prompting the customer for input.
 *
 * To create a Bank and open it for business issue the command
 * <code>java Bank</code>.
 *
 * @see BankAccount
 * @version 7
 */

public class Bank
{
    private String bankName;           // the name of this Bank
    private Terminal atm;              // for talking with the customer
    private int balance = 0;           // total cash on hand
    private int transactionCount = 0;  // number of Bank transactions
    private Month month;               // the current month.
    private Map accountList;           // mapping names to accounts.

    private int checkFee = 2;            // cost for each check
    private int transactionFee = 1;      // fee for each transaction
    private int monthlyCharge = 5;       // monthly charge
    private double interestRate = 0.05;  // annual rate paid on savings
    private int maxFreeTransactions = 3; // for savings accounts

    // what the banker can ask of the bank

    private static final String BANKER_COMMANDS =
    "Banker commands: " +
        "exit, open, customer, nextmonth, report, help.";

    // what the customer can ask of the bank

    private static final String CUSTOMER_TRANSACTIONS =
    "    Customer transactions: " +
    "deposit, withdraw, transfer, balance, cash check, quit, help.";

    /**
     * Construct a Bank with the given name and Terminal.
     *
     * @param bankName the name for this Bank.
     * @param atm  this Bank's Terminal.
     */
```



```
    public Bank( String bankName, Terminal atm )
    {
        this.atm      = atm;
        this.bankName = bankName;
        accountList   = new TreeMap();
        month         = new Month();
    }

    /**
     * Simulates interaction with a Bank.
     * Presents the user with an interactive loop, prompting for
     * banker transactions and in the case of the banker
     * transaction "customer", an account id and further
     * customer transactions.
     */

    public void visit()
    {
        instructUser();

        String command;
        while (!(command =
                 atm.readWord("banker command: ")).equals("exit")) {

            if (command.startsWith("h")) {
                help( BANKER_COMMANDS );
            }
            else if (command.startsWith("o")) {
                openNewAccount();
            }
            else if (command.startsWith("n")) {
                newMonth();
            }
            else if (command.startsWith("r")) {
                report();
            }
            else if (command.startsWith( "c" ) ) {
                BankAccount acct = whichAccount();
                if ( acct != null ) {
                        processTransactionsForAccount( acct );
                }
            }
            else {
                // Unrecognized Request
                atm.println( "unknown command: " + command );
            }
        }
        report();
        atm.println( "Goodbye from " + bankName );
    }


    // Open a new bank account,
    // prompting the user for information.

    private void openNewAccount()
```

```
113      {
114          String accountName = atm.readWord("Account name: ");
115          char accountType =
116              atm.readChar( "Type of account (r/c/f/s): " );
117          try {
118              int startup = readPosAmt( "Initial deposit: " );
119              BankAccount newAccount;
120              switch( accountType ) {
121              case 'c':
122                  newAccount = new CheckingAccount(startup, this);
123                  break;
124              case 'f':
125                  newAccount = new FeeAccount(startup, this);
126                  break;
127              case 's':
128                  newAccount = new SavingsAccount(startup, this);
129                  break;
130              case 'r':
131                  newAccount = new RegularAccount( startup, this );
132                  break;
133              default:
134                  atm.println("invalid account type: " + accountType);
135                  return;
136              }
137              accountList.put( accountName, newAccount );
138              atm.println( "opened new account " + accountName
139                           + " with $" + startup );
140          } // end of try block
141          catch (NegativeAmountException e) {
142              atm.errPrintln(
143                "can't start with a negative balance");
144          }
145          catch (InsufficientFundsException e) {
146              atm.errPrintln("Initial deposit less than fee");
147          }
148      }
149
150      // Prompt the customer for transaction to process.
151      // Then send an appropriate message to the account.
152
153      private void processTransactionsForAccount( BankAccount acct )
154      {
155          help( CUSTOMER_TRANSACTIONS );
156
157          String transaction;
158          while (!(transaction =
159                   atm.readWord("    transaction: ")).equals("quit")) {
160
161              try {
162                  if ( transaction.startsWith( "h" ) ) {
163                      help( CUSTOMER_TRANSACTIONS );
164                  }
165                  else if ( transaction.startsWith( "d" ) ) {
166                      int amount = readPosAmt( "    amount:" );
167                      atm.println("    deposited "
168                                  + acct.deposit( amount ));
```



```
169                  }
170                  else if ( transaction.startsWith( "w" ) ) {
171                      int amount = readPosAmt( "    amount:" );
172                      atm.println("    withdrew "
173                                  + acct.withdraw( amount ));
174                  }
175                  else if ( transaction.startsWith( "c" ) ) {
176                      int amount = readPosAmt( "    amount of check: " );
177                      try { // to cast acct to CheckingAccount ...
178                          atm.println("    cashed check for " +
179                             ((CheckingAccount) acct).honorCheck( amount ))
180                      }
181                      catch (ClassCastException e) {
182                          // if not a checking account, report error
183                          atm.errPrintln(
184                              "    Sorry, not a checking account." );
185                      }
186                  }
187                  else if (transaction.startsWith("t")) {
188                      atm.print( "    to ");
189                      BankAccount toacct = whichAccount();
190                      if (toacct != null) {
191                      int amount = readPosAmt("    amount to transfer: ");
192                      atm.println("    transfered "
193                          + toacct.deposit(acct.withdraw(amount)));
194                      }
195                  }
196                  else if (transaction.startsWith("b")) {
197                      atm.println("    current balance "
198                                  + acct.requestBalance());
199                  }
200                  else {
201                      atm.println("    sorry, unknown transaction" );
202                  }
203              }
204              catch (InsufficientFundsException e) {
205                  atm.errPrintln( "    Insufficient funds " +
206                                  e.getMessage() );
207              }
208              catch (NegativeAmountException e) {
209                atm.errPrintln("    Sorry, negative amounts disallowed." );
210              }
211              atm.println();
212          }
213      }
214
215      // Prompt for an account name (or number), look it up
216      // in the account list. If it's there, return it;
217      // otherwise report an error and return null.
218
219      private BankAccount whichAccount()
220      {
221          String accountName = atm.readWord( "account name: " );
222          BankAccount account = (BankAccount) accountList.get(accountName);
223          if (account == null) {
224              atm.println( "not a valid account" );
```

```
225          }
226          return account;
227      }
228
229      // Action to take when a new month starts.
230      // Update the month field by sending a next message.
231      // Loop on all accounts, sending each a newMonth message.
232
233      private void newMonth()
234      {
235          month.next();
236          Iterator i = accountList.keySet().iterator();
237          while ( i.hasNext() ) {
238              String name = (String) i.next();
239              BankAccount acct = (BankAccount)accountList.get(name);
240              try {
241                  acct.newMonth();
242              }
243              catch (InsufficientFundsException exception) {
244                  atm.errPrintln(
245                     "Insufficient funds in account \"" +
246                     name + "\" for monthly fee" );
247              }
248          }
249      }
250
251      // Report bank activity. For each BankAccount,
252      // print the customer id (name or number), balance, and
253      // the number of transactions. Then print Bank totals.
254
255      private void report()
256      {
257          atm.println( bankName + " report for " + month );
258          atm.println( "\nSummaries of individual accounts:" );
259          atm.println( "account  balance   transaction count" );
260          for (Iterator i = accountList.keySet().iterator();
261               i.hasNext(); ) {
262              String accountName = (String) i.next();
263              BankAccount acct = (BankAccount) accountList.get(accountName)
264              atm.println(accountName + "\t$" + acct.getBalance() + "\t\t"
265                          + acct.getTransactionCount());
266          }
267          atm.println( "\nBank totals");
268          atm.println( "open accounts: " + getNumberOfAccounts() );
269          atm.println( "cash on hand: $" + getBalance());
270          atm.println( "transactions:  " + getTransactionCount());
271          atm.println();
272      }
273
274
275      // Welcome the user to the bank and instruct her on
276      //  her options.
277
278      private void instructUser()
279      {
280          atm.println( "Welcome to " + bankName );
```



```
281          atm.println( month.toString() );
282          atm.println( "Open some accounts and work with them." );
283          help( BANKER_COMMANDS );
284      }
285
286      // Display a help string.
287
288      private void help( String helpString )
289      {
290          atm.println( helpString );
291          atm.println();
292      }
293
294      // Read amount prompted for from the atm.
295      // Throw a NegativeAmountException if amount < 0
296
297      private int readPosAmt( String prompt )
298           throws NegativeAmountException
299      {
300          int amount = atm.readInt( prompt );
301          if (amount < 0) {
302              throw new NegativeAmountException();
303          }
304          return amount;
305      }
306
307      /**
308       * Increment bank balance by given amount.
309       *
310       * @param amount the amount increment.
311       */
312
313      public void incrementBalance(int amount)
314      {
315          balance += amount;
316      }
317
318      /**
319       * Increment by one the count of transactions,
320       * for this bank.
321       */
322
323      public void countTransaction()
324      {
325          transactionCount++;
326      }
327
328      /**
329       * Get the number of transactions performed by this bank.
330       *
331       * @return number of transactions performed.
332       */
333
334      public int getTransactionCount( )
335      {
336          return transactionCount ;
```

```
337      }
338
339      /**
340       * The charge this bank levies for cashing a check.
341       *
342       * @return check fee
343       */
344
345      public int getCheckFee( )
346      {
347          return checkFee ;
348      }
349
350      /**
351       * The charge this bank levies for a transaction.
352       *
353       * @return the transaction fee
354       */
355
356      public int getTransactionFee( )
357      {
358          return transactionFee ;
359      }
360
361      /**
362       * The charge this bank levies each month.
363       *
364       * @return the monthly charge
365       */
366
367      public int getMonthlyCharge( )
368      {
369          return monthlyCharge;
370      }
371
372      /**
373       * The current interest rate on savings.
374       *
375       * @return the interest rate
376       */
377
378      public double getInterestRate( )
379      {
380          return interestRate;
381      }
382
383      /**
384       * The number of free transactions per month.
385       *
386       * @return the number of transactions
387       */
388
389      public int getMaxFreeTransactions()
390      {
391          return maxFreeTransactions;
392      }
```



```
393
394      /**
395       * Get the current bank balance.
396       *
397       * @return current bank balance.
398       */
399
400      public int getBalance()
401      {
402          return balance;
403      }
404
405      /**
406       * Get the current number of open accounts.
407       *
408       * @return number of open accounts.
409       */
410
411      public int getNumberOfAccounts()
412      {
413          return accountList.size();
414      }
415
416      /**
417       * Run the simulation by creating and then visiting a new Bank.
418       * <p>
419       * A -e argument causes the input to be echoed.
420       * This can be useful for executing the program against
421       * a test script, e.g.,
422       * <pre>
423       *   java Bank -e < Bank.in
424       * </pre>
425       *
426       * @param args the command line arguments:
427       *         <pre>
428       *         -e echo input.
429       *         bankName any other command line argument.
430       *         </pre>
431       */
432
433      public static void main( String[] args )
434      {
435          // parse the command line arguments for the echo
436          // flag and the name of the bank
437
438          boolean echo    = false;         // default does not echo
439          String bankName = "River Bank";  // default bank name
440
441          for (int i = 0; i < args.length; i++ ) {
442              if (args[i].equals("-e")) {
443                  echo = true;
444              }
445              else {
446                  bankName = args[i];
447              }
448          }
```

```
449           Bank aBank = new Bank( bankName, new Terminal(echo) );
450           aBank.visit();
451       }
452   }
```