```
 1  // joi/1/bank/BankAccount.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * A BankAccount object has a private field to keep track
 8   * of this account's current balance, and public methods to
 9   * return and change the balance.
10   *
11   * @see Bank
12   * @version 1
13   */
14
15  public class BankAccount
16  {
17      private int balance;  // work only in whole dollars
18
19      /**
20       * A constructor for creating a new bank account.
21       *
22       * @param initialBalance the opening balance.
23       */
24
25      public BankAccount( int initialBalance )
26      {
27          this.deposit( initialBalance );
28      }
29
30      /**
31       * Withdraw the amount requested.
32       *
33       * @param amount the amount to be withdrawn.
34       */
35
36      public void withdraw( int amount )
37      {
38          balance = balance - amount;
39      }
40
41      /**
42       * Deposit the amount requested.
43       *
44       * @param amount the amount to be deposited.
45       */
46
47      public void deposit( int amount )
48      {
49          balance = balance + amount;
50      }
51
52      /**
53       * The current account balance.
54       *
55       * @return the current balance.
56       */
```



```
57
58      public int getBalance()
59      {
60          return balance;
61      }
62  }
```

```
 1  // joi/1/bank/Bank.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * A Bank object simulates the behavior of a simple bank/ATM.
 8   * It contains a Terminal object and two BankAccount objects.
 9   *
10   * Its single public method is open, which opens the this Bank
11   * for business, prompting the customer for input.
12   *
13   * To create a Bank and open it for business issue the command
14   * <code>java Bank</code>.
15   *
16   * @see BankAccount
17   * @version 1
18   */
19
20  public class Bank
21  {
22      private String bankName;      // the name of this Bank
23
24      private Terminal atm;         // for talking with the customer
25
26      private BankAccount account1; // two accounts to play with
27      private BankAccount account2;
28
29      private static final int INITIAL_BALANCE = 200;
30      private static final String HELPSTRING =
31          "Transactions: exit, help, deposit, withdraw, balance";
32
33      /**
34       * Construct a Bank with the given name.
35       * Create two new BankAccounts, each with a starting balance
36       * of initialBalance.
37       *
38       * @param name the name of the Bank.
39       */
40
41      public Bank( String name )
42      {
43          bankName = name;
44          atm      = new Terminal();
45          account1 = new BankAccount( INITIAL_BALANCE );
46          account2 = new BankAccount( INITIAL_BALANCE );
47      }
48
49      /**
50       * Open the Bank for business.
51       *
52       * Send a whichAccount message prompting for a BankAccount
53       * number, then send a processTransactionsForAccount
54       * message to do the work.
55       */
56
```



```
57      public void open()
58      {
59          atm.println( "Welcome to " + bankName );
60          boolean bankIsOpen = true;
61          while ( bankIsOpen ) {
62              BankAccount account = this.whichAccount();
63              if ( account == null ) {
64                  bankIsOpen = false;
65              }
66              else {
67                  this.processTransactionsForAccount(account);
68              }
69          }
70          atm.println( "Goodbye from " + bankName );
71      }
72
73      // Prompt the user for an account number and return the
74      // corresponding BankAccount object. Return null when
75      // the Bank is about to close.
76
77      private BankAccount whichAccount()
78      {
79          int  accountNumber =
80              atm.readInt("Account number (1 or 2), 0 to shut down: ");
81
82          if ( accountNumber == 1 ) {
83              return account1;
84          }
85          else if ( accountNumber == 2 ) {
86              return account2;
87          }
88          else if ( accountNumber == 0 ) {
89              return null;
90          }
91          else {
92              atm.println( "No account numbered " +
93                                  accountNumber + "; try again" );
94              return this.whichAccount();
95          }
96      }
97
98      // Prompt the user for transaction to process.
99      // Then send an appropriate message to account.
100
101     private void processTransactionsForAccount( BankAccount account)
102     {
103         atm.println( HELPSTRING );
104
105         boolean moreTransactions = true;
106         while ( moreTransactions ) {
107             String command = atm.readWord( "transaction: " );
108             if ( command.equals( "exit" ) ) {
109                 moreTransactions = false;
110             }
111             else if ( command.equals( "help" ) ) {
112                 atm.println( HELPSTRING );
```

```
113             }
114             else if ( command.equals( "deposit" ) ) {
115                 int amount = atm.readInt( "amount: " );
116                 account.deposit( amount );
117             }
118             else if ( command.equals( "withdraw" ) ) {
119                 int amount = atm.readInt( "amount: " );
120                 account.withdraw( amount );
121             }
122             else if ( command.equals( "balance" ) ) {
123                 atm.println( account.getBalance() );
124             }
125             else{
126                 atm.println("sorry, unknown transaction" );
127             }
128         }
129     }
130
131     /**
132      * The Bank simulation program begins here when the user
133      * issues the command <code>java Bank</code>.
134
135      * @param args the command line arguments (ignored).
136      */
137
138     public static void main( String[] args )
139     {
140         Bank javaBank = new Bank( "Engulf and Devour" );
141         javaBank.open();
142     }
143 }
```