

ATM Case Study Implementation

This section contains the ATM system's complete working implementation.

- ATM
- Screen
- Keypad
- CashDispenser
- DepositSlot
- Account
- BankDatabase
- Transaction
- BalanceInquiry
- Withdrawal
- Deposit

Class ATM represents the ATM as a whole. Lines 5–11 implement the class's attributes. We determine all but one of these attributes from the UML class diagrams of Figs. 26.9–26.10. Line 5 declares the bool attribute `userAuthenticated` from Fig. 26.10. Line 6 declares an attribute not found in our UML design—int attribute `currentAccountNumber`, which keeps track of the account number of the current authenticated user. Lines 7–11 declare reference-type instance variables corresponding to the ATM class's associations modeled in the class diagram of Fig. 26.9. These attributes allow the ATM to access its parts (i.e., its Screen, Keypad, CashDispenser and DepositSlot) and interact with the bank's account information database (i.e., a BankDatabase object). Lines 14–20 declare an enumeration that corresponds to the four options in the ATM's main menu (i.e., balance inquiry, withdrawal, deposit and exit).

Lines 23–32 declare class ATM's constructor, which initializes the class's attributes. When an ATM object is first created, no user is authenticated, so line 25 initializes `userAuthenticated` to false. Line 26 initializes `currentAccountNumber` to 0 because there is no current user yet. Lines 27–30 instantiate new objects to represent the parts of the ATM. Recall that class ATM has composition relationships with classes Screen, Keypad, CashDispenser and DepositSlot, so class ATM is responsible for their creation. Line 31 creates a new BankDatabase. As you'll soon see, the BankDatabase creates two Account objects that can be used to test the ATM. [Note: If this were a real ATM system, the ATM class would receive a reference to an existing database object created by the bank. However, in this implementation, we are only simulating the bank's database, so class ATM creates the BankDatabase object with which it interacts.]

```

// ATM.cs

2 // Represents an automated teller machine.

3 public class ATM
4 {
5     private bool userAuthenticated; // true if user is authenticated
6     private int currentAccountNumber; // user's account number
7     private Screen screen; // reference to ATM's screen
8     private Keypad keypad; // reference to ATM's keypad
9     private CashDispenser cashDispenser; // ref to ATM's cash dispenser
10    private DepositSlot depositSlot; // reference to ATM's deposit slot
11    private BankDatabase bankDatabase; // ref to account info database
12
13    // enumeration that represents main menu options
14    private enum MenuOption
15    {
16        BALANCE_INQUIRY = 1,
17        WITHDRAWAL = 2,
18        DEPOSIT = 3,
19        EXIT_ATM = 4
20    }
21
22    // parameterless constructor initializes instance variables
23    public ATM()
24    {
25        userAuthenticated = false; // user is not authenticated to start
26        currentAccountNumber = 0; // no current account number to start
27        screen = new Screen(); // create screen
28        keypad = new Keypad(); // create keypad
29        cashDispenser = new CashDispenser(); // create cash dispenser
30        depositSlot = new DepositSlot(); // create deposit slot

```

```
31 bankDatabase = new BankDatabase(); // create account info database
32 }
33
34 // start ATM
35 public void Run()
36 {
37 // welcome and authenticate users; perform transactions
38 while (true) // infinite loop
39 {
40 // loop while user is not yet authenticated
41 while (!userAuthenticated)
42 {
43 screen.DisplayMessageLine("\nWelcome!");
44 AuthenticateUser(); // authenticate user
45 }
46
47 PerformTransactions(); // for authenticated user
48 userAuthenticated = false; // reset before next ATM session
49 currentAccountNumber = 0; // reset before next ATM session
50 screen.DisplayMessageLine("\nThank you! Goodbye!");
51 }
52 }
53
54 // attempt to authenticate user against database
55 private void AuthenticateUser()
56 {
57 // prompt for account number and input it from user
58 screen.DisplayMessage("\nPlease enter your account number: ");
59 int accountNumber = keypad.GetInput();
60
```

```
61 // prompt for PIN and input it from user
62 screen.DisplayMessage("\nEnter your PIN: ");
63 int pin = keypad.GetInput();
64
65 // set userAuthenticated to boolean value returned by database
66 userAuthenticated =
67 bankDatabase.AuthenticateUser(accountNumber, pin);
// check whether authentication succeeded
70 if (userAuthenticated)
71 currentAccountNumber = accountNumber; // save user's account #
72 else
73 screen.DisplayMessageLine(
74 "Invalid account number or PIN. Please try again.");
75 }
76
77 // display the main menu and perform transactions
78 private void PerformTransactions()
79 {
80 Transaction currentTransaction; // transaction being processed
81 bool userExited = false; // user has not chosen to exit
82
83 // loop while user has not chosen exit option
84 while (!userExited)
85 {
86 // show main menu and get user selection
87 int mainMenuSelection = DisplayMainMenu();
88
89 // decide how to proceed based on user's menu selection
90 switch ((MenuOption) mainMenuSelection)
91 {
```

```

92 // user chooses to perform one of three transaction types
93 case MenuOption.BALANCE_INQUIRY:
94 case MenuOption.WITHDRAWAL:
95 case MenuOption.DEPOSIT:
96 // initialize as new object of chosen type
97 currentTransaction =
98 CreateTransaction(mainMenuSelection);
99 currentTransaction.Execute(); // execute transaction
100 break;
101 case MenuOption.EXIT_ATM: // user chose to terminate session
102 screen.DisplayMessageLine("\nExiting the system...");
103 userExited = true; // this ATM session should end
104 break;
105 default: // user did not enter an integer from 1-4
106 screen.DisplayMessageLine(
107 "\nYou did not enter a valid selection. Try again.");
108 break;
109 }
110 }
111 }
112
113 // display the main menu and return an input selection
114 private int DisplayMainMenu()
115 {
116 screen.DisplayMessageLine("\nMain menu:");
117 screen.DisplayMessageLine("1 - View my balance");
118 screen.DisplayMessageLine("2 - Withdraw cash");
119 screen.DisplayMessageLine("3 - Deposit funds");
120 screen.DisplayMessageLine("4 - Exit\n");
121 screen.DisplayMessage("Enter a choice: ");

```

```
return keypad.GetInput(); // return user's selection
123 }
124
125 // return object of specified Transaction derived class
126 private Transaction CreateTransaction(int type)
127 {
128     Transaction temp = null; // null Transaction reference
129
130     // determine which type of Transaction to create
131     switch ((MenuOption) type)
132     {
133         // create new BalanceInquiry transaction
134         case MenuOption.BALANCE_INQUIRY:
135             temp = new BalanceInquiry(currentAccountNumber,
136 screen, bankDatabase);
137             break;
138         case MenuOption.WITHDRAWAL: // create new Withdrawal transaction
139             temp = new Withdrawal(currentAccountNumber, screen,
140 bankDatabase, keypad, cashDispenser);
141             break;
142         case MenuOption.DEPOSIT: // create new Deposit transaction
143             temp = new Deposit(currentAccountNumber, screen,
144 bankDatabase, keypad, depositSlot);
145             break;
146     }
147
148     return temp;
149 }
150 }
```

Implementing the Operation

The class diagram of Fig. 26.10 does not list any operations for class ATM. We now implement one operation (i.e., public method) in class ATM that allows an external client of the

class (i.e., class ATMCaseStudy; Section 26.4.12) to tell the ATM to run. ATM method Run

(Fig. 26.13, lines 35–52) uses an infinite loop (lines 38–51) to repeatedly welcome a user, attempt to authenticate the user and, if authentication succeeds, allow the user to perform transactions. After an authenticated user performs the desired transactions and exits, the ATM resets itself, displays a goodbye message and restarts the process for the next user. We use an infinite loop here to simulate the fact that an ATM appears to run continuously until the bank turns it off (an action beyond the user's control). An ATM user can exit the system, but cannot turn off the ATM completely.

Authenticating the User

We refer to the requirements document to determine the steps necessary to authenticate the user before allowing transactions to occur. Line 58 of method AuthenticateUser invokes method DisplayMessage of the ATM's screen to prompt the user to enter an account

number. Line 59 invokes method GetInput of the ATM's keypad to obtain the user's input,

then stores this integer in local variable accountNumber. Method AuthenticateUser next prompts the user to enter a PIN (line 62), and stores the PIN in local variable pin (line 63). Next, lines 66–67 attempt to authenticate the user by passing the accountNumber and pin entered by the user to the bankDatabase's AuthenticateUser method.

Class Screen

In this class screen we will display the amount in Indian currency for how is deposited or how much is credited to account using atm.

```
// Screen.cs
```

```
2 // Represents the screen of the ATM
```

```
3 using System;
```

```
4
```

```
5 public class Screen
```

```

6 {
7 // displays a message without a terminating carriage return
8 public void DisplayMessage(string message)
9 {
10 Console.Write(message);
11 }
12
13 // display a message with a terminating carriage return
14 public void DisplayMessageLine(string message)
15 {
16 Console.WriteLine(message);
17 }
18
19 // display a dollar amount
20 public void DisplayRupeeAmount(decimal amount)
21 {
22 Console.Write("{0:C}", amount);
23 }
24 }

```

Class Keypad

Class Keypad (Fig. 26.15) represents the keypad of the ATM and is responsible for receiving all user input. Recall that we are simulating this hardware, so we use the computer's

keyboard to approximate the keypad. We use method `Console.ReadLine` to obtain keyboard input from the user. A computer keyboard contains many keys not found on the ATM's keypad. We assume that the user presses only the keys on the computer keyboard that also appear on the keypad—the keys numbered 0–9 and the Enter key.

```

// Keypad.cs
2 // Represents the keypad of the ATM.
3 using System;

```



```

4
5 public class Keypad
6 {
7 // return an integer value entered by user
8 public int GetInput()
9 {
10 return int.Parse(Console.ReadLine());
11 }
12 }

```

Class Cash Dispenser

Class CashDispenser represents the cash dispenser of the ATM. Line 6 declares constant INITIAL_COUNT, which indicates the number of 20 bills in the cash dispenser when the ATM starts (i.e., 500). Line 7 implements attribute billCount

, which keeps track of the number of bills remaining in the CashDispenser

at any time. The constructor (lines 10–13) sets billCount to the initial count. [Note: We assume that the process of adding more bills to the CashDispenser and updating the billCount occur outside the ATM system.] Class CashDispenser has two public methods—

DispenseCash (lines 16–21) and IsSufficientCashAvailable (lines 24–31). The class trusts that a client (i.e., Withdrawal) calls method DispenseCash only after establishing that sufficient cash is available by calling method IsSufficientCashAvailable. Thus, DispenseCash simulates dispensing the requested amount of cash without checking whether sufficient cash is available.

```

1 // CashDispenser.cs
2 // Represents the cash dispenser of the ATM
3 public class CashDispenser
4 {
5 // the default initial number of bills in the cash dispenser
6 private const int INITIAL_COUNT = 500;

```

```

7 private int billCount; // number of $20 bills remaining
8
9 // parameterless constructor initializes billCount to INITIAL_COUNT
10 public CashDispenser()
11 {
12     billCount = INITIAL_COUNT; // set billCount to INITIAL_COUNT
13 }
14
15 // simulates dispensing the specified amount of cash
16 public void DispenseCash(decimal amount)
17 {
18     // number of $20 bills required
19     int billsRequired = ((int) amount) / 20;
20     billCount -= billsRequired;
21 }
22
23 // indicates whether cash dispenser can dispense desired amount
24 public bool IsSufficientCashAvailable(decimal amount)
25 {
26     // number of $20 bills required
27     int billsRequired = ((int) amount) / 20;
28
29     // return whether there are enough bills available
30     return (billCount >= billsRequired);
31 }
32 }

```

Class DepositSlot

Class `DepositSlot` represents the deposit slot of the ATM. This class simulates the functionality of a real hardware deposit slot. `DepositSlot` has no attributes and only one method—`IsDepositEnvelopeReceived` (lines 7–10)—which indicates whether

a deposit envelope was received.

```
// DepositSlot.cs
2 // Represents the deposit slot of the ATM
3 public class DepositSlot
4 {
5 // indicates whether envelope was received (always returns true,
6 // because this is only a software simulation of a real deposit slot)
7 public bool IsDepositEnvelopeReceived()
8 {
9 return true; // deposit envelope was received
10 }
11 }
```

Class Account

Class Account (Fig. 26.18) represents a bank account. Each Account has four attributes (modeled in Fig. 26.10)—`accountNumber`, `pin`, `availableBalance` and `totalBalance`.

Lines 5–8 implement these attributes as private instance variables. For each of the instance

variables `accountNumber`, `availableBalance` and `totalBalance`, we provide a property with the same name as the attribute, but starting with a capital letter. For example, property

`AccountNumber` corresponds to the `accountNumber` attribute modeled in Fig. 26.10. Clients

of this class do not need to modify the `accountNumber` instance variable, so `AccountNumber`

is declared as a read-only property (i.e., it provides only a get accessor).

```
1 // DepositSlot.cs
2 // Represents the deposit slot of the ATM
3 public class DepositSlot
4 {
5 // indicates whether envelope was received (always returns true,
6 // because this is only a software simulation of a real deposit slot)
```

```

7 public bool IsDepositEnvelopeReceived()
8 {
9     return true; // deposit envelope was received
10 }
11 }

```

Fig. 26.17 | Class DepositSlot represents the ATM's deposit slot.

```

1 // Account.cs
2 // Class Account represents a bank account.
3 public class Account
4 {
5     private int accountNumber; // account number
6     private int pin; // PIN for authentication
7     private decimal availableBalance; // available withdrawal amount
8     private decimal totalBalance; // funds available + pending deposit
9
10    // four-parameter constructor initializes attributes
11    public Account(int theAccountNumber, int thePIN,
12        decimal theAvailableBalance, decimal theTotalBalance)
13    {
14        accountNumber = theAccountNumber;
15        pin = thePIN;
16        availableBalance = theAvailableBalance;
17        totalBalance = theTotalBalance;
18    }
19
20    // read-only property that gets the account number
21    public int AccountNumber
22    {
23        get
24        {

```

```
25 return accountNumber;
26 }
27 }
28
29 // read-only property that gets the available balance
30 public decimal AvailableBalance
31 {
32     get
33     {
34         return availableBalance;
35     }
36 }
37
38 // read-only property that gets the total balance
39 public decimal TotalBalance
40 {
41     get
42     {
43         return totalBalance;
44     }
45 }
46
47 // determines whether a user-specified PIN matches PIN in Account
48 public bool ValidatePIN(int userPIN)
49 {
50     return (userPIN == pin);
51 }
52
53 // credits the account (funds have not yet cleared)
54 public void Credit(decimal amount)
```

```

55 {
56 totalBalance += amount; // add to total balance
57 }
58
59 // debits the account
60 public void Debit(decimal amount)
61 {
62 availableBalance -= amount; // subtract from available balance
63 totalBalance -= amount; // subtract from total balance
64 }
65 }

```

Class Withdrawal

Class Withdrawal extends Transaction and represents an ATM withdrawal transaction. This class expands on the “skeleton” code for this class developed in

```

1 // BalanceInquiry.cs
2 // Represents a balance inquiry ATM transaction
3 public class BalanceInquiry : Transaction
4 {
5 // five-parameter constructor initializes base class variables
6 public BalanceInquiry(int userAccountNumber,
7 Screen atmScreen, BankDatabase atmBankDatabase)
8 : base(userAccountNumber, atmScreen, atmBankDatabase) { }
9
10 // performs transaction; overrides Transaction's abstract method
11 public override void Execute()
12 {
13 // get the available balance for the current user's Account
14 decimal availableBalance =
15 Database.GetAvailableBalance(AccountNumber);

```

```

16
17 // get the total balance for the current user's Account
18 decimal totalBalance = Database.GetTotalBalance(AccountNumber);
19
20 // display the balance information on the screen
21 UserScreen.DisplayMessageLine("\nBalance Information:");
22 UserScreen.DisplayMessage(" - Available balance: ");
23 UserScreen.DisplayDollarAmount(availableBalance);
24 UserScreen.DisplayMessage("\n - Total balance: ");
25 UserScreen.DisplayDollarAmount(totalBalance);
26 UserScreen.DisplayMessageLine("");
27 }
28 }

```

Class Withdrawal

Recall from the class diagram of that class Withdrawal has one attribute, amount, which line 5 declares as a decimal instance variable. models as sociations between class Withdrawal and classes Keypad and CashDispenser, for which

lines 6–7 implement reference attributes keypad and cashDispenser, respectively. Line 10 declares a constant corresponding to the cancel menu option.

```

1 // Withdrawal.cs
2 // Class Withdrawal represents an ATM withdrawal transaction.
3 public class Withdrawal : Transaction
4 {
5     private decimal amount; // amount to withdraw
6     private Keypad keypad; // reference to Keypad
7     private CashDispenser cashDispenser; // reference to cash dispenser
8
9     // constant that corresponds to menu option to cancel
10    private const int CANCELED = 6;
11
12    // five-parameter constructor

```

```
13 public Withdrawal(int userAccountNumber, Screen atmScreen,
14 BankDatabase atmBankDatabase, Keypad atmKeypad,
15 CashDispenser atmCashDispenser)
16 : base(userAccountNumber, atmScreen, atmBankDatabase)
17 {
18 // initialize references to keypad and cash dispenser
19 keypad = atmKeypad;
20 cashDispenser = atmCashDispenser;
21 }
22
23 // perform transaction, overrides Transaction's abstract method
24 public override void Execute()
25 {
26 bool cashDispensed = false; // cash was not dispensed yet
27
28 // transaction was not canceled yet
29 bool transactionCanceled = false;
30
31 // loop until cash is dispensed or the user cancels
32 do
33 {
34 // obtain the chosen withdrawal amount from the user
35 int selection = DisplayMenuOfAmounts();
36
37 // check whether user chose a withdrawal amount or canceled
38 if (selection != CANCELED)
39 {
40 // set amount to the selected dollar amount
41 amount = selection;
42
```



```
43 // get available balance of account involved
44 decimal availableBalance =
45 Database.GetAvailableBalance(AccountNumber);
46
47 // check whether the user has enough money in the account
48 if (amount <= availableBalance)
49 {
50 // check whether the cash dispenser has enough money
51 if (cashDispenser.IsSufficientCashAvailable(amount))
52 {
53 // debit the account to reflect the withdrawal
54 Database.Debit(AccountNumber, amount);
55
56 cashDispenser.DispenseCash(amount); // dispense cash
57 cashDispensed = true; // cash was dispensed
58
59 // instruct user to take cash
60 UserScreen.DisplayMessageLine(
61 "\nPlease take your cash from the cash dispenser.");
62 }
63 else // cash dispenser does not have enough cash
64 UserScreen.DisplayMessageLine(
65 "\nInsufficient cash available in the ATM." +
66 "\n\nPlease choose a smaller amount.");
67 }
68 else // not enough money available in user's account
69 UserScreen.DisplayMessageLine(
70 "\nInsufficient cash available in your account." +
71 "\n\nPlease choose a smaller amount.");
72 }
```

```

73 else
74 {
75 UserScreen.DisplayMessageLine("\nCanceling transaction...");
76 transactionCanceled = true; // user canceled the transaction
77 }
78 } while ((!cashDispensed) && (!transactionCanceled));
79 }
80
81 // display a menu of withdrawal amounts and the option to cancel;
82 // return the chosen amount or 6 if the user chooses to cancel
83 private int DisplayMenuOfAmounts()
84 {
85 int userChoice = 0; // variable to store return value
86
87 // array of amounts to correspond to menu numbers
88 int[] amounts = { 0, 100, 200, 500, 1000 };
89
90 // loop while no valid choice has been made
91 while (userChoice == 0)
92 {
93 // display the menu
94 UserScreen.DisplayMessageLine("\nWithdrawal options:");
95 UserScreen.DisplayMessageLine("1 - 100");
96 UserScreen.DisplayMessageLine("2 - 200");
97 UserScreen.DisplayMessageLine("3 - 500");
98 UserScreen.DisplayMessageLine("4 - 2000");

```

Class Withdrawal's constructor (lines 13–21) has five parameters. It uses the constructor initializer to pass parameters userAccountNumber, atmScreen and atmBankDatabase to base class Transaction's constructor to set the attributes that Withdrawal inherits

from Transaction. The constructor also takes references atmKeypad and atmCashDispenser as parameters and assigns them to reference-type attributes keypad and cashDispenser, respectively.

Overriding abstract Method Execute

Class Withdrawal overrides Transaction's abstract method Execute with a concrete implementation (lines 24–79) that performs the steps involved in a withdrawal. Line 26 declares and initializes a local bool variable cashDispensed. This variable indicates whether

cash has been dispensed (i.e., whether the transaction has completed successfully) and is initially false. Line 29 declares and initializes to false a bool variable transactionCanceled to indicate that the transaction has not yet been canceled by the user.

Lines 32–78 contain a do...while statement that executes its body until cash is dispensed (i.e., until cashDispensed becomes true) or until the user chooses to cancel (i.e., until transactionCanceled becomes true). We use this loop to continuously return the user to the start of the transaction if an error occurs (i.e., the requested withdrawal amount is greater than the user's available balance or greater than the amount of cash in the cash

```
100 UserScreen.DisplayMessageLine("6 - Cancel transaction");
101 UserScreen.DisplayMessage(
102 "\nChoose a withdrawal option (1-6): ");
103
104 // get user input through keypad
105 int input = keypad.GetInput();
106
107 // determine how to proceed based on the input value
108 switch (input)
109 {
110 // if the user chose a withdrawal amount (i.e., option
111 // 1, 2, 3, 4, or 5), return the corresponding amount
112 // from the amounts array
113 case 1: case 2: case 3: case 4: case 5:
114 userChoice = amounts[ input ]; // save user's choice
115 break;
```

```

116 case CANCELED: // the user chose to cancel
117 userChoice = CANCELED; // save user's choice
118 break;
119 default:
120 UserScreen.DisplayMessageLine(
121 "\nInvalid selection. Try again.");
122 break;
123 }
124 }
125
126 return userChoice;
127 }
128 }

```

Class Deposit

Class Deposit inherits from Transaction and represents an ATM deposit transaction. Recall from the class diagram of that class Deposit has one attribute, amount, which line 5 declares as a decimal instance variable. Lines 6–7 create reference attributes keypad and depositSlot that implement the associations between class Deposit and classes Keypad and DepositSlot, modeled in Line 10 declares a constant CANCELED that corresponds to the value a user enters to cancel a deposit transaction.

```

1 // Deposit.cs
2 // Represents a deposit ATM transaction.
3 public class Deposit : Transaction
4 {
5     private decimal amount; // amount to deposit
6     private Keypad keypad; // reference to the Keypad
7     private DepositSlot depositSlot; // reference to the deposit slot
8
9     // constant representing cancel option
10    private const int CANCELED = 0;

```

```
11
12 // five-parameter constructor initializes class's instance variables
13 public Deposit(int userAccountNumber, Screen atmScreen,
14 BankDatabase atmBankDatabase, Keypad atmKeypad,
15 DepositSlot atmDepositSlot)
16 : base(userAccountNumber, atmScreen, atmBankDatabase)
17 {
18 // initialize references to keypad and deposit slot
19 keypad = atmKeypad;
20 depositSlot = atmDepositSlot;
21 } -parameter constructor
22
23 // perform transaction; overrides Transaction's abstract method
24 public override void Execute()
25 {
26 amount = PromptForDepositAmount(); // get deposit amount from user
27
28 // check whether user entered a deposit amount or canceled
29 if (amount != CANCELED)
30 {
31 // request deposit envelope containing specified amount
32 UserScreen.DisplayMessage(
33 "\nPlease insert a deposit envelope containing ");
34 UserScreen.DisplayDollarAmount(amount);
35 UserScreen.DisplayMessageLine(" in the deposit slot.");
36
37 // retrieve deposit envelope
38 bool envelopeReceived = depositSlot.IsDepositEnvelopeReceived();
39
40 // check whether deposit envelope was received
```

```
41 if (envelopeReceived)
42 {
43 UserScreen.DisplayMessageLine(
44 "\nYour envelope has been received.\n" +
45 "The money just deposited will not be available " +
46 "until we \nverify the amount of any " +
47 "enclosed cash, and any enclosed checks clear.");
48
49 // credit account to reflect the deposit
50 Database.Credit(AccountNumber, amount);
51 }
52 else
53 UserScreen.DisplayMessageLine(
54 "\nYou did not insert an envelope, so the ATM has " +
55 "canceled your transaction.");
56 }
57 else
58 UserScreen.DisplayMessageLine("\nCanceling transaction...");
59 }
60
61 // prompt user to enter a deposit amount to credit
62 private decimal PromptForDepositAmount()
63 {
64 // display the prompt and receive input
65 UserScreen.DisplayMessage(
66 "\nPlease input a deposit amount in Indian Rupee(or 0 to cancel): ");
67 int input = keypad.GetInput();
68
69 // check whether the user canceled or entered a valid amount
70 if (input == CANCELED)
```

```
71 return CANCELED;
72 else
73 return input;
74 }
75 }
```

Method Execute (lines 24–59) overrides abstract method Execute in base class Transaction with a concrete implementation that performs the steps required in a deposit transaction. Line 26 prompts the user to enter a deposit amount by invoking private utility method PromptForDepositAmount (declared in lines 62–74) and sets attribute amount to the value returned. Method PromptForDepositAmount asks the user to enter a deposit amount as an integer number of cents (because the ATM’s keypad does not contain a decimal point; this is consistent with many real ATMs) and returns the decimal value representing the dollar amount to be deposited.

Class ATMCaseStudy

Class ATMCaseStudy simply allows us to start, or “turn on,” the ATM and test the implementation of our ATM system model. Class ATMCaseStudy’s Main method (lines 6–10) simply instantiates a new ATM object named theATM (line 8) and invokes its Run

method (line 9) to start the ATM

```
.
1 // ATMCaseStudy.cs
2 // App for testing the ATM case study.
3 public class ATMCaseStudy
4 {
5 // Main method is the app's entry point
6 public static void Main(string[] args)
7 {
8 ATM theATM = new ATM();
9 theATM.Run();
10 }
11 }
```

Wrap-Up

In this chapter, you used inheritance to tune the design of the ATM software system, and you fully implemented the ATM in C#. Congratulations on completing the entire ATM case study! We hope you found this experience to be valuable and that it reinforced many of the object-oriented programming concepts that you've learned.

```
1 Account.cs
2 // Class Account represents a bank account.
3 public class Account
4 {
5     private int accountNumber; // account number
6     private int pin; // PIN for authentication
7
8     // automatic read-only property AvailableBalance
9     public decimal AvailableBalance { get; private set; }
10
11     // automatic read-only property TotalBalance
12     public decimal TotalBalance { get; private set; }
13
14     // parameterless constructor
15     public Account()
16     {
17         // constructor body code
18     }
19
20     // validates user PIN
21     public bool ValidatePIN()
22     {
23         // ValidatePIN method body code
24     }
25
```



```

26 // credits the account
27 public void Credit()
28 {
29 // Credit method body code
30 }
31
32 // debits the account
33 public void Debit()
34 {

```

26.5 False. The UML requires that we italicize abstract class names and operation names.

26.6 The design for class Transaction yields the code in Fig. 26.26. In the implementation, a

constructor initializes private instance variables userScreen and database to actual objects, and read-only properties UserScreen and Database access these instance variables. These properties allow classes derived from Transaction to access the ATM's screen and interact with the bank's database. We chose the names of the UserScreen and Database properties for clarity—we wanted to

avoid property names that are the same as the class names Screen and BankDatabase, which can be confusing.

```

35 // Debit method body code
36 }
37 }

1 // Fig. 26.26: Transaction.cs
2 // Abstract base class Transaction represents an ATM transaction.
3 public abstract class Transaction
4 {
5 private int accountNumber; // indicates account involved
6 private Screen userScreen; // ATM's screen
7 private BankDatabase database; // account info database
8
9 // parameterless constructor
10 public Transaction()

```

```
11 {
12 // constructor body code
13 }
14
15 // read-only property that gets the account number
16 public int AccountNumber
17 {
18 get
19 {
20 return accountNumber;
21 }
22 }
23
24 // read-only property that gets the screen reference
25 public Screen UserScreen
26 {
27 get
28 {
29 return userScreen;
30 }
31 }
32
33 // read-only property that gets the bank database reference
34 public BankDatabase Database
35 {
36 get
37 {
38 return database;
39 }
40 }
```

41

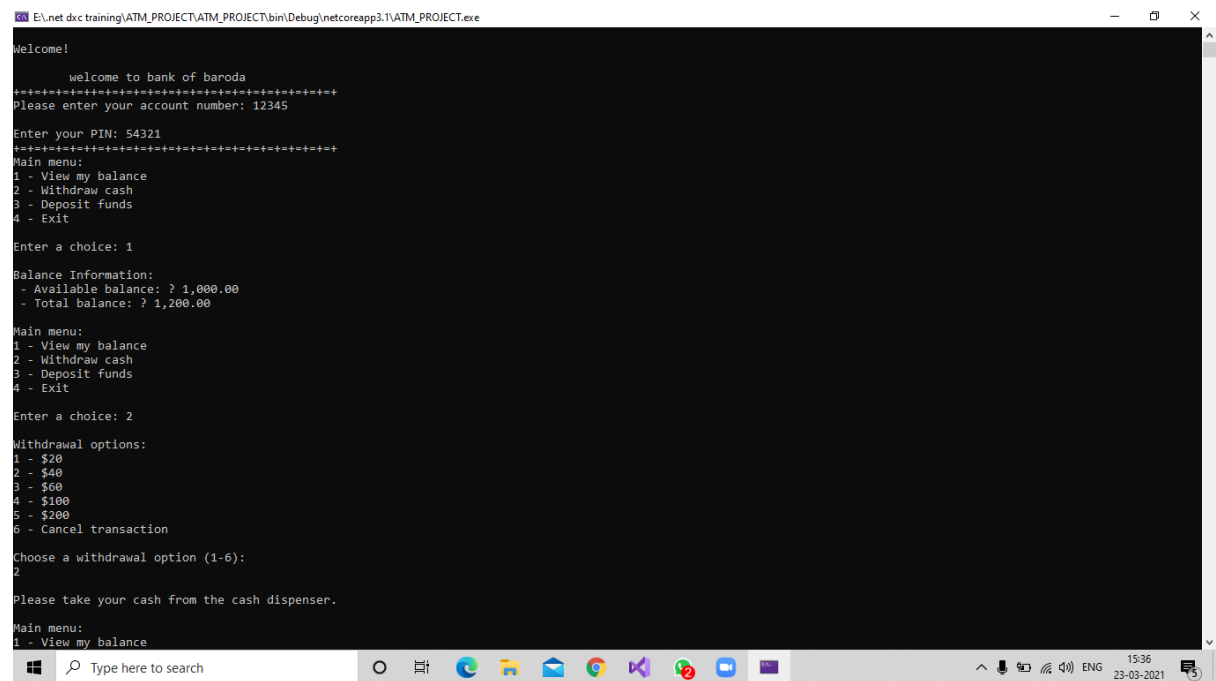
42 // perform the transaction (overridden by each derived class)

43 public abstract void Execute();

44 }

OUTPUT

1.



```

E:\net dxc training\ATM_PROJECT\ATM_PROJECT\bin\Debug\netcoreapp3.1\ATM_PROJECT.exe
Welcome!

      welcome to bank of baroda
=====
Please enter your account number: 12345

Enter your PIN: 54321
=====
Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice: 1

Balance Information:
- Available balance: ? 1,000.00
- Total balance: ? 1,200.00

Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit

Enter a choice: 2

Withdrawal options:
1 - $20
2 - $40
3 - $60
4 - $100
5 - $200
6 - Cancel transaction

Choose a withdrawal option (1-6):
2

Please take your cash from the cash dispenser.

Main menu:
1 - View my balance
```

2.

```
E:\net dxc training\ATM_PROJECT\ATM_PROJECT\bin\Debug\netcoreapp3.1\ATM_PROJECT.exe
4 - Exit
Enter a choice: 2
Withdrawal options:
1 - $20
2 - $40
3 - $60
4 - $100
5 - $200
6 - Cancel transaction
Choose a withdrawal option (1-6):
2
Please take your cash from the cash dispenser.
Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit
Enter a choice: 1
Balance Information:
- Available balance: ? 900.00
- Total balance: ? 1,160.00
Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit
Enter a choice: 3
Please input a deposit amount in dollars (or 0 to cancel): 5000
Please insert a deposit envelope containing ? 50.00 in the deposit slot.
Your envelope has been received.
The money just deposited will not be available until we
verify the amount of any enclosed cash, and any enclosed checks clear.
```

3.

```
E:\net dxc training\ATM_PROJECT\ATM_PROJECT\bin\Debug\netcoreapp3.1\ATM_PROJECT.exe
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit
Enter a choice: 3
Please input a deposit amount in dollars (or 0 to cancel): 5000
Please insert a deposit envelope containing ? 50.00 in the deposit slot.
Your envelope has been received.
The money just deposited will not be available until we
verify the amount of any enclosed cash, and any enclosed checks clear.
Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit
Enter a choice: 1
Balance Information:
- Available balance: ? 1,010.00
- Total balance: ? 1,210.00
Main menu:
1 - View my balance
2 - Withdraw cash
3 - Deposit funds
4 - Exit
Enter a choice: 4
Exiting the system...
Thank you! Goodbye!
Welcome!
welcome to bank of baroda
*****
Please enter your account number:
```