A lengthy statement can be spread over several lines. If a single statement must be split across lines, choose breaking points that make sense, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines until the end of the statement.

Figure 2.16 shows the precedence of the operators introduced in this chapter. The operators are shown from top to bottom in decreasing order of precedence. All these operators, with the exception of the assignment operator, =, associate from left to right. Addition is left associative, so an expression like x + y + z is evaluated as if it had been written as (x + y) + z. The assignment operator, =, associates from right to left, so an expression like x = y = 0 is evaluated as if it had been written as x = (y = 0), which, as we will soon see, first assigns the value 0 to variable y and then assigns the result of that assignment, 0, to x.

Refer to the operator precedence chart (see the complete chart in Appendix A) when writing expressions containing many operators. Confirm that the operations in the expression are performed in the order you expect. If you are uncertain about the order of evaluation in a complex expression, use parentheses to force the order, exactly as you would do in algebraic expressions. Observe that some operators, such as assignment, =, associate from right to left rather than from left to right.

2.9 (Optional) Software Engineering Case Study: Examining the Requirements Document

Now we begin our optional object-oriented design and implementation case study. The "Software Engineering Case Study" sections at the ends of this and the next several chapters will ease you into object orientation by examining an automated teller machine (ATM) case study. This case study will provide you with a concise, carefully paced, complete design and implementation experience. In Chapters 3–8 and 10, we will perform the various steps of an object-oriented design (OOD) process using the UML while relating these steps to the object-oriented concepts discussed in the chapters. Appendix J implements the ATM using the techniques of object-oriented programming (OOP) in Java. We present the complete case-study solution. This is not an exercise; rather, it is an end-to-end learning experience that concludes with a detailed walkthrough of the Java code that implements our design. It will acquaint you with the kinds of substantial problems encountered in industry and their potential solutions. We hope you enjoy this learning experience.

* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Fig. 2.16 | Precedence and associativity of operations discussed.

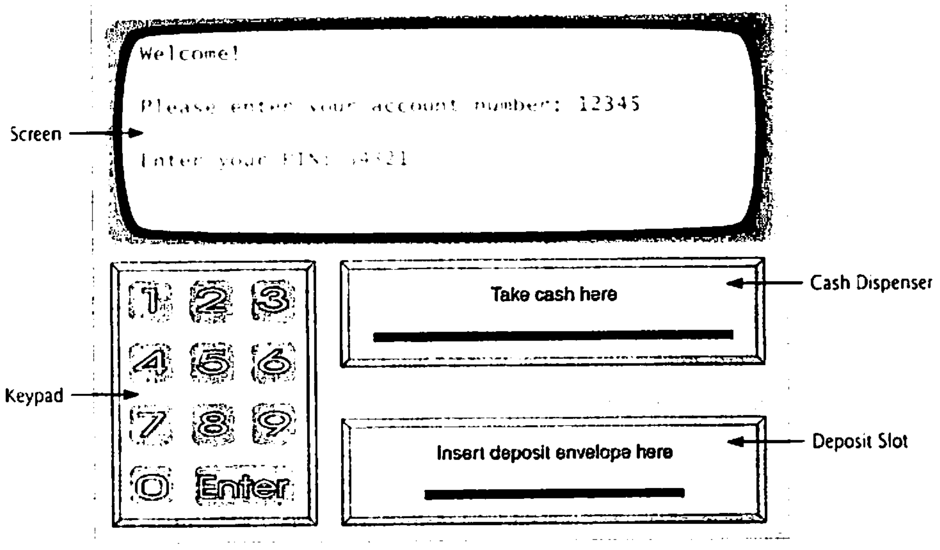
We begin our design process by presenting a requirements document that specifies the overall purpose of the ATM system and *what* it must do. Throughout the case study, we refer to the requirements document to determine precisely what functionality the system must include.

Requirements Document
A local bank intends to install a new automated teller machine (ATM) to allow users (i.e., bank customers) to perform basic financial transactions (Fig. 2.17). Each user can have only one account at the bank. ATM users should be able to view their account balance, withdraw cash (i.e., take money out of an account) and deposit funds (i.e., place money into an account).

The user interface of the automated teller machine contains the following components:

- a screen that displays messages to the user
- a keypad that receives numeric input from the user
- a cash dispenser that dispenses cash to the user and
- a deposit slot that receives deposit envelopes from the user.

The cash dispenser begins each day loaded with 500 $20 bills. [*Note:* Due to the limited scope of this case study, certain elements of the ATM described here do not accurately mimic those of a real ATM. For example, a real ATM typically contains a device that reads a user's account number from an ATM card, whereas this ATM asks the user to type an account number on the keypad. A real ATM also usually prints a receipt at the end of a session, but all output from this ATM appears on the screen.]


```
Welcome!

Please enter your account number: 12345

Enter your PIN: 54321
```

Fig. 2.17 | Automated teller machine user interface.

The bank wants you to develop software to perform the financial transactions initiated by bank customers through the ATM. The bank will integrate the software with the ATM's hardware at a later time. The software should encapsulate the functionality of the hardware devices (e.g., cash dispenser, deposit slot) within software components, but it need not concern itself with how these devices perform their duties. The ATM hardware has not been developed yet, so instead of writing your software to run on the ATM, you should develop a first version of the software to run on a personal computer. This version should use the computer's monitor to simulate the ATM's screen, and the computer's keyboard to simulate the ATM's keypad.

An ATM session consists of authenticating a user (i.e., proving the user's identity) based on an account number and personal identification number (PIN), followed by creating and executing financial transactions. To authenticate a user and perform transactions, the ATM must interact with the bank's account information database (i.e., an organized collection of data stored on a computer). For each bank account, the database stores an account number, a PIN and a balance indicating the amount of money in the account. [*Note:* We assume that the bank plans to build only one ATM, so we do not need to worry about multiple ATMs accessing this database at the same time. Furthermore, we assume that the bank does not make any changes to the information in the database while a user is accessing the ATM. Also, any business system like an ATM faces reasonably complicated security issues that go well beyond the scope of a first- or second-semester computer science course. We make the simplifying assumption, however, that the bank trusts the ATM to access and manipulate the information in the database without significant security measures.]

Upon first approaching the ATM (assuming no one is currently using it), the user should experience the following sequence of events (shown in Fig. 2.17):

1. The screen displays a welcome message and prompts the user to enter an account number.
2. The user inputs a five-digit account number using the keypad.
3. The screen prompts the user to enter the PIN (personal identification number) associated with the specified account number.
4. The user enters a five-digit PIN using the keypad.
5. If the user enters a valid account number and the correct PIN for that account, the screen displays the main menu (Fig. 2.18). If the user enters an invalid account number or an incorrect PIN, the screen displays an appropriate message, then the ATM returns to *Step 1* to restart the authentication process.

After the ATM authenticates the user, the main menu (Fig. 2.18) should contain a numbered option for each of the three types of transactions: balance inquiry (option 1), withdrawal (option 2) and deposit (option 3). The main menu also should contain an option to allow the user to exit the system (option 4). The user then chooses either to perform a transaction (by entering 1, 2 or 3) or to exit the system (by entering 4).

If the user enters 1 to make a balance inquiry, the screen displays the user's account balance. To do so, the ATM must retrieve the balance from the bank's database.

The following steps describe the actions that occur when the user enters 2 to make a withdrawal:

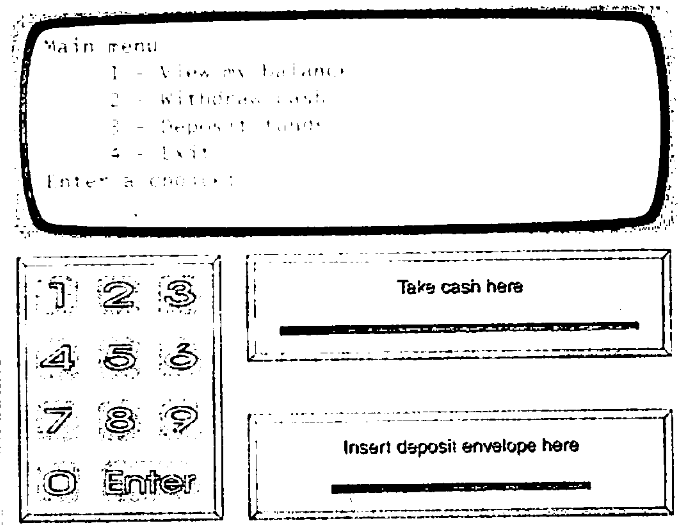
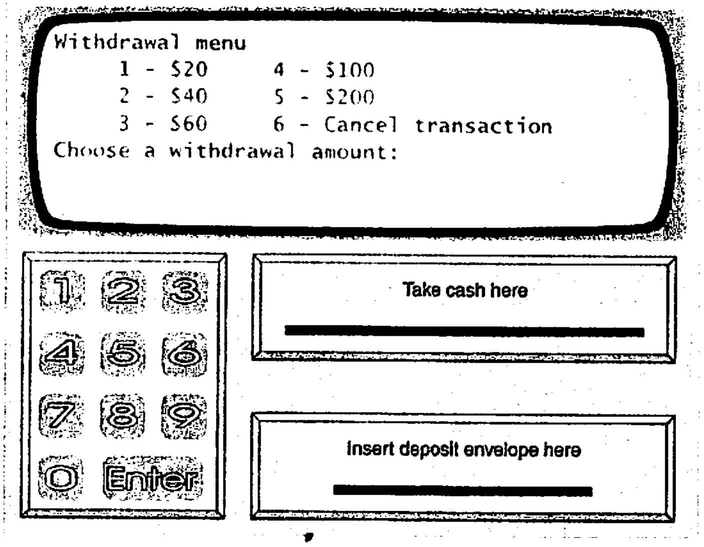


Fig. 2.18 | ATM main menu.

1. The screen displays a menu (shown in Fig. 2.19) containing standard withdrawal amounts: $20 (option 1), $40 (option 2), $60 (option 3), $100 (option 4) and $200 (option 5). The menu also contains an option to allow the user to cancel the transaction (option 6).
2. The user inputs a menu selection using the keypad.
3. If the withdrawal amount chosen is greater than the user's account balance, the screen displays a message stating this and telling the user to choose a smaller amount. The ATM then returns to *Step 1*. If the withdrawal amount chosen is less than or equal to the user's account balance (i.e., an acceptable amount), the ATM proceeds to *Step 4*. If the user chooses to cancel the transaction (option 6), the ATM displays the main menu and waits for user input.
4. If the cash dispenser contains enough cash to satisfy the request, the ATM proceeds to *Step 5*. Otherwise, the screen displays a message indicating the problem and telling the user to choose a smaller withdrawal amount. The ATM then returns to *Step 1*.
5. The ATM debits the withdrawal amount from the user's account in the bank's database (i.e., subtracts the withdrawal amount from the user's account balance).
6. The cash dispenser dispenses the desired amount of money to the user.
7. The screen displays a message reminding the user to take the money.

The following steps describe the actions that occur when the user enters 3 to make a deposit:



```
Withdrawal menu
     1 - $20      4 - $100
     2 - $40      5 - $200
     3 - $60      6 - Cancel transaction
Choose a withdrawal amount:
```

Fig. 2.19 | ATM withdrawal menu.

1. The screen prompts the user to enter a deposit amount or type 0 (zero) to cancel the transaction.
2. The user inputs a deposit amount or 0 using the keypad. [*Note:* The keypad does not contain a decimal point or a dollar sign, so the user cannot type a real dollar amount (e.g., $1.25). Instead, the user must enter a deposit amount as a number of cents (e.g., 125). The ATM then divides this number by 100 to obtain a number representing a dollar amount (e.g., 125 ÷ 100 = 1.25).]
3. If the user specifies a deposit amount, the ATM proceeds to *Step 4*. If the user chooses to cancel the transaction (by entering 0), the ATM displays the main menu and waits for user input.
4. The screen displays a message telling the user to insert a deposit envelope into the deposit slot.
5. If the deposit slot receives a deposit envelope within two minutes, the ATM credits the deposit amount to the user's account in the bank's database (i.e., adds the deposit amount to the user's account balance). [*Note:* This money is not immediately available for withdrawal. The bank first must physically verify the amount of cash in the deposit envelope, and any checks in the envelope must clear (i.e., money must be transferred from the check writer's account to the check recipient's account). When either of these events occur, the bank appropriately updates the user's balance stored in its database. This occurs independently of the ATM system.] If the deposit slot does not receive a deposit envelope within this time period, the screen displays a message that the system has canceled the transaction due to inactivity. The ATM then displays the main menu and waits for user input.

After the system successfully executes a transaction, the system should return to the main menu so that the user can perform additional transactions. If the user chooses to exit the system, the screen should display a thank you message, then display the welcome message for the next user.

Analyzing the ATM System

The preceding statement is a simplified example of a requirements document. Typically, such a document is the result of a detailed process of requirements gathering that might include interviews with possible users of the system and specialists in fields related to the system. For example, a systems analyst who is hired to prepare a requirements document for banking software (e.g., the ATM system described here) might interview financial experts to gain a better understanding of what the software must do. The analyst would use the information gained to compile a list of system requirements to guide systems designers as they design the system.

The process of requirements gathering is a key task of the first stage of the software life cycle. The software life cycle specifies the stages through which software goes from the time it is first conceived to the time it is retired from use. These stages typically include: analysis, design, implementation, testing and debugging, deployment, maintenance and retirement. Several software life cycle models exist, each with its own preferences and specifications for when and how often software engineers should perform each of these stages. Waterfall models perform each stage once in succession, whereas iterative models may repeat one or more stages several times throughout a product's life cycle.

The analysis stage of the software life cycle focuses on defining the problem to be solved. When designing any system, one must *solve the problem right*, but of equal importance, one must *solve the right problem*. Systems analysts collect the requirements that indicate the specific problem to solve. Our requirements document describes the requirements of our ATM system in sufficient detail that you do not need to go through an extensive analysis stage—it has been done for you.

To capture what a proposed system should do, developers often employ a technique known as use case modeling. This process identifies the use cases of the system, each of which represents a different capability that the system provides to its clients. For example, ATMs typically have several use cases, such as "View Account Balance," "Withdraw Cash," "Deposit Funds," "Transfer Funds Between Accounts" and "Buy Postage Stamps." The simplified ATM system we build in this case study allows only the first three use cases.

Each use case describes a typical scenario for which the user uses the system. You have already read descriptions of the ATM system's use cases in the requirements document; the lists of steps required to perform each transaction type (i.e., balance inquiry, withdrawal and deposit) actually described the three use cases of our ATM—"View Account Balance," "Withdraw Cash" and "Deposit Funds."

Use Case Diagrams

We now introduce the first of several UML diagrams in the case study. We create a use case diagram to model the interactions between a system's clients (in this case study, bank customers) and its use cases. The goal is to show the kinds of interactions users have with a system without providing the details—these are provided in other UML diagrams (which we present throughout this case study). Use case diagrams are often accompanied by informal text that describes the use cases in more detail—like the text that appears in the requirements document. Use case diagrams are produced during the analysis stage of the software life cycle. In larger systems, use case diagrams are indispensable tools that help system designers remain focused on satisfying the users' needs.

Figure 2.20 shows the use case diagram for our ATM system. The stick figure represents an actor, which defines the roles that an external entity—such as a person or another system—plays when interacting with the system. For our automated teller machine, the actor is a User who can view an account balance, withdraw cash and deposit funds from the ATM. The User is not an actual person, but instead comprises the roles that a real person—when playing the part of a User—can play while interacting with the ATM. Note that a use case diagram can include multiple actors. For example, the use case diagram for a real bank's ATM system might also include an actor named Administrator that refills the cash dispenser each day.

Our requirements document supplies the actors—"ATM users should be able to view their account balance, withdraw cash and deposit funds." Therefore, the actor in each of the three use cases is the user who interacts with the ATM. An external entity—a real person—plays the part of the user to perform financial transactions. Figure 2.20 shows one actor, whose name, User, appears below the actor in the diagram. The UML models each use case as an oval connected to an actor with a solid line.

Software engineers (more precisely, systems designers) must analyze the requirements document or a set of use cases and design the system before programmers implement it in a particular programming language. During the analysis stage, systems designers focus on understanding the requirements document to produce a high-level specification that describes *what* the system is supposed to do. The output of the design stage—a design specification—should specify clearly *how* the system should be constructed to satisfy these requirements. In the next several "Software Engineering Case Study" sections, we perform the steps of a simple object-oriented design (OOD) process on the ATM system to produce a design specification containing a collection of UML diagrams and supporting text. The UML is designed for use with any OOD process. Many such processes exist, the most well-known of which is the Rational Unified Process™ (RUP) developed by Rational Software Corporation. RUP is a rich process intended for designing "industrial strength" applications. For this case study, we present our own simplified design process, designed for students in first- and second-semester programming courses.

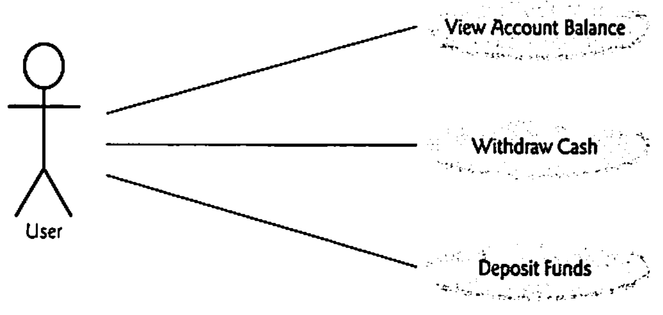


Fig. 2.20 | Use case diagram for the ATM system from the user's perspective.

Designing the ATM System

We now begin the design stage of our ATM system. A *system* is a set of components that interact to solve a problem. For example, to perform the ATM system's designated tasks, our ATM system has a user interface (Fig. 2.17), contains software that executes financial transactions and interacts with a database of bank-account information. *System structure* describes the system's objects and their interrelationships. *System behavior* describes how the system changes as its objects interact with one another. Every system has both structure and behavior—designers must specify both. There are several distinct types of system structures and behaviors. For example, the interactions among objects in the system differ from those between the user and the system, yet both constitute a portion of the system behavior.

The UML 2 specifies 13 diagram types for documenting the models of systems. Each models a distinct characteristic of a system's structure or behavior—six diagrams relate to system structure; the remaining seven relate to system behavior. We list here only the six types used in our case study—one of these (class diagrams) models system structure, whereas the remaining five model system behavior. We overview the remaining seven UML diagram types in Appendix L, UML 2: Additional Diagram Types.

1. *Use case diagrams*, such as the one in Fig. 2.20, model the interactions between a system and its external entities (actors) in terms of use cases (system capabilities, such as "View Account Balance," "Withdraw Cash" and "Deposit Funds").

2. *Class diagrams*, which you will study in Section 3.10, model the classes, or "building blocks," used in a system. Each noun or "thing" described in the requirements document is a candidate to be a class in the system (e.g., `Account`, `Keypad`). Class diagrams help us specify the structural relationships between parts of the system. For example, the ATM system class diagram will specify that the ATM is physically composed of a screen, a keypad, a cash dispenser and a deposit slot.

3. *State machine diagrams*, which you will study in Section 5.11, model the ways in which an object changes state. An object's *state* is indicated by the values of all the object's attributes at a given time. When an object changes state, that object may behave differently in the system. For example, after validating a user's PIN, the ATM transitions from the "user not authenticated" state to the "user authenticated" state, at which point the ATM allows the user to perform financial transactions (e.g., view account balance, withdraw cash, deposit funds).

4. *Activity diagrams*, which you will also study in Section 5.11, model an object's *activity*—the object's workflow (sequence of events) during program execution. An activity diagram models the actions the object performs and specifies the order in which the object performs these actions. For example, an activity diagram shows that the ATM must obtain the balance of the user's account (from the bank's account information database) before the screen can display the balance to the user.

5. *Communication diagrams* (called *collaboration diagrams* in earlier versions of the UML) model the interactions among objects in a system, with an emphasis on *what* interactions occur. You will learn in Section 7.14 that these diagrams show which objects must interact to perform an ATM transaction. For example, the ATM must communicate with the bank's account information database to retrieve an account balance.

6. Sequence diagrams also model the interactions among the objects in a system, but unlike communication diagrams, they emphasize *when* interactions occur. You will learn in Section 7.14 that these diagrams help show the order in which interactions occur in executing a financial transaction. For example, the screen prompts the user to enter a withdrawal amount before cash is dispensed.

In Section 3.10, we continue designing our ATM system by identifying the classes from the requirements document. We accomplish this by extracting key nouns and noun phrases from the requirements document. Using these classes, we develop our first draft of the class diagram that models the structure of our ATM system.

Internet and Web Resources

The following URLs provide information on object-oriented design with the UML.

`www-306.ibm.com/software/rational/uml/`

Lists frequently asked questions about the UML, provided by IBM Rational.

`www.softdocwiz.com/Dictionary.htm`

Hosts the Unified Modeling Language Dictionary, which lists and defines all terms used in the UML.

`www-306.ibm.com/software/rational/offerings/design.html`

Provides information about IBM Rational software available for designing systems. Provides downloads of 30-day trial versions of several products, such as IBM Rational Rose® XDE Developer.

`www.embarcadero.com/products/describe/index.html`

Provides a free 15-day license to download a trial version of Describe™—a UML modeling tool from Embarcadero Technologies®.

`www.borland.com/together/index.html`

Provides a free 30-day license to download a trial version of Borland® Together® ControlCenter™—a software-development tool that supports the UML.

`www.ilogix.com/rhapsody/rhapsody.cfm`

Provides a free 30-day license to download a trial version of I-Logix Rhapsody®—a UML 2 based model-driven development environment.

`argouml.tigris.org`

Contains information and downloads for ArgoUML, a free open-source UML tool written in Java.

`www.objectsbydesign.com/books/booklist.html`

Lists books on the UML and object-oriented design.

`www.objectsbydesign.com/tools/umltools_byCompany.html`

Lists software tools that use the UML, such as IBM Rational Rose, Embarcadero Describe, Sparx Systems Enterprise Architect, I-Logix Rhapsody and Gentleware Poseidon for UML.

`www.ootips.org/ood-principles.html`

Provides answers to the question, "What Makes a Good Object-Oriented Design?"

`www.parlezuml.com/tutorials/java/class/index_files/frame.htm`

Provides a UML tutorial for Java developers that presents UML diagrams side by side with the Java code that implements them.

`www.cetus-links.org/oo_uml.html`

Introduces the UML and provides links to numerous UML resources.

`www.agilemodeling.com/essays/umlDiagrams.htm`

Provides in-depth descriptions and tutorials on each of the 13 UML 2 diagram types.

Recommended Readings

The following books provide information on object-oriented design with the UML.