2

Messages and Objects Everywhere

THE SMALLTALK-80 LANGUAGE

> ***When I use a word, . . .***
> ***it means just what I choose it to mean***
> ***— neither more nor less.***
> LEWIS CARROLL, *Alice's Adventures in Wonderland*

Smalltalk strives to use a small number of consistent abstractions and terms, unconstrained by conventions or terms from other programming languages. For example, a Smalltalk procedure or subroutine is called a "method." We will try to justify the Smalltalk names as soon as you leam enough to understand the excuse, but for now we will keep the unconventional names to a minimum in this tutorial. Generally we will give the Smalltalk name with the conventional name in parentheses, or vice versa. Variable names in Smalltalk are often highly descriptive, and thus quite long. Following the Smalltalk tradition. we use capitals as visual separators (moveTower) instead of hyphens (move-tower). In the next few paragraphs we explain Smalltalk relying mainly on Pascal terminology, with four figures highlighting different aspects of the moveTower procedure.

The "moveTower" procedure has local names for its arguments, and these are underlined below. All text that is between double quotes is a comment (comments are shown in italics in this chapter only). It is surprising that four languages have four different notations for something as simple as a comment. Next is an expression, (height > 0), that evaluates to true or false. The Smalltalk if-statement is like Pascal except that the Boolean expression precedes, instead of follows, the "if." Next is a block, similar to the C curly bracket notation, except that Smalltalk surrounds blocks with square brackets. There is a comment at the end of the procedure.

```
moveTower: height from: fromPin to: toPin using: usingPin
   "Recursive procedure to move the disk at a height from one
   pinto another pin using a third pin"
   (height > 0) ifTrue: [
      self moveTower: (height-1) from: fromPin to: usingPin using: toPin,
      self moveDisk: fromPin to: toPin.
      self moveTower: (height-1) from: usingPin to: toPin using: fromPin]


   "This comment gives an example of how to run this program. Select
   the following and choose 'doit' from the middle-button menu.
      (Object new) moveTower: 3 from: 1 to: 3 using: 2
```

In Smalltalk, periods are used to separate statements. Pascal and C programmers will find this syntax familiar provided they remember to use '.' instead of';'. There are three statements in the block, shown in boxes below, and they are executed sequentially.

```
moveTower: height from: fromPin to: toPin using: usingPin
   "Recursive procedure to move the disk at a height from one
   pin to another pin using a third pin"
   (height > 0) ifTrue: [

      | self moveTower: (height-1) from: fromPin to: usingPin using: toPirT|
      j self moveDisk: fromPin to: toPin.1
      | self moveTower: (height-1) from: usingPin to: toPin using: fromPin |]


   "This comment gives an example of how to run this program. Select
   the following and choose 'do it' from the middle-button menu.
      (Object new) moveTower: 3 from: 1 to: 3 using: 2
```

The designers of the Smalltalk language chose a format for procedure names that encourages the programmer to describe each of the arguments. The idea is to provide more than just the order of the arguments to help the programmer remember which one is which. Each part of a procedure name ends with a colon and is followed by the argument it describes. This notation could be used in any language.

The four underlined words in the first line below are the four parts of the name of the procedure (method) that is being defined. When you want to talk about the procedure, squeeze all the parts of its name together; in this example, the actual name of the procedure we have been working with is moveTower:from:to:using:. It has four arguments and corresponds to the Pascal procedure named "movetower." This procedure calls three procedures (itself twice and moveDisk:to: once) and those names are also underlined. The interleaving of pieces of procedure names with the arguments is just syntactic sugar, and you may find it useful to translate calls on this procedure to the familiar format of procedure name followed by arguments:

moveTower:from:to:using: (height, fromPin, toPin, usingPin).

```
moveTower: height from: fromPin to: toPin using: usingPin
    "Recursive procedure to move the disk at a height from one
    pin to another pin using a third pin"
    (height > 0) ifTrue: [
        self moveTower: (height -1) from: fromPin to: usingPin using: toPin.
        self moveDisk: fromPin to: toPin.
        self moveTower: (height -1) from: usingPin to: toPin using: fromPin]

    "This comment gives an example of how to run this program. Select
    the following and choose 'do it' from the middle-button menu.
        (Object new) moveTower: 3 from: 1 to: 3 using: 2
```

As mentioned above, Smalltalk strives for consistent abstractions. While most languages treat operators and procedure names as separate entities, Smalltalk lumps them together, calling them "message selectors." Both > and moveTower:from:to:using: are message selectors.

You've probably heard that Smalltalk is an object-oriented system, and may be curious to know what an "object" is. An object is a package of data and procedures that belong together. Specifically, all constants and the contents of all variables are objects. An object in Smalltalk is like a record in Pascal, but much richer and more versatile. Below, we have underlined all the objects in our example. The only things that don't denote objects are the message selectors (operators or procedure names), the comments, and a few punctuation characters.

```
moveTower: height from: fromPin to: toPin using: usingPin
    "Recursive procedure to move the disk at a height from one
    pin to another pin using a third pin"
    (height > 0) ifTrue: [
        self moveTower: (height-1) from: fromPin to: usingPin using: toPin.
        self moveDisk: fromPin to: toPin.
        self moveTower: (height -1) from: usingPin to: toPin using: fromPin]

    "This comment gives an example of how to run this program. Select
    the following and choose 'do it' from the middle-button menu.
        (Object new) moveTower: 3 from: 1 to: 3 using: 2
```

Most systems get work done in a variety of ways: by calling procedures, applying operators to operands, conditionally executing blocks, and so forth. Following the goal of using a small number of consistent abstractions, Smalltalk has exactly one way of getting work done: by "sending messages" to objects. A message is just an operator or procedure name (message selector) with its operands. The object that receives a message, the "receiver," appears just to the left of the message. We have boxed some of the messages in the code below.

```
moveTower: height from: fromPin to: toPin using: usingPin
    "Recursive procedure to move the disk at a height from one
    pin to another pin using a third pin"
    (height F^OI) ifTrue: [
        self | moveTower: (height I -11) from: fromPin to: usingPin using: toPin].
        self I moveDisk: fromPin to: toPinI._____
        self | moveTower: (height I -Tj) from: usingPin to: toPin using: fromPin ]

    "This comment gives an example of how to run this program. Select
    the following and choose 'do it' from the middle-button menu.
        (Object new) moveTower: 3 from: 1 to: 3 using: 2
```

Smalltalk always returns a value as the result of each procedure (method), and, as you might expect from an object-oriented language, that result is also an object. For example, height-1 returns an integer and height > 0 returns a boolean.

You now know enough that we can explain more Smalltalk lingo. The terms "method" (procedure) and "selector" (procedure name) come from the question, "How do we *select* the *method* an object will use to respond to this message?" The answer is, "Use the *selector* to find the right *method* to execute." A message is just the operator or procedure name (message selector) along with its arguments. "Calling a proce-

dure" is translated in Smalltalkese as "sending a message." From now on, we will use the term "method" instead of "procedure" or "subroutine."

If you talk to yourself while you read code (don't be bashful, everyone does), then you need to know how to "talk" Smalltalk, height > 0 does exactly what you think it does, and you can pronounce it just the way you would in other languages ("height is greater than zero"); but it is really shorthand for "the object height receives the message greater-than with the argument zero." For the really dedicated code talkers, see Appendix 2. We will sprinkle Smalltalkese throughout this tutorial, but you can survive this experience without learning the complete dialect.

The object-message paradigm is natural for simulation programs. For example, sending the message throttleOpen: 30 to the object that is simulating an automobile engine might mean that the gas pedal is pressed to 30 percent of maximum. When an object receives a message, it looks up the message name to see if it understands the message. If the message is found, it starts executing the "method" that tells how to respond to the message.

Just as a Pascal procedure may call other procedures, a method may need to call other methods. The way to start another method is to send a message to an object. Sometimes you want to send a message to the same object that received the current message. How is that object named locally? In other words, when a Smalltalk object talks to itself, what does it call itself? Why, "self," naturally! Not surprisingly, messages to self are common. You can see them sprinkled throughout the program on the previous page. How does Smalltalk handle recursion? In Pascal, the definition of a procedure can include a call on itself. In Smalltalk, the code within a method sends a mesage to the object self, in particular, a message with the same selector as the current method.

Specifying an object, sending it a message, and getting back another object as the result are the only things that ever happen in Smalltalk code. Things that require new kinds of constructs in other languages, such as control structures and arithmetic operators, are simply messages sent to objects in Smalltalk. The result of one message can be used as the object that receives another message, or as an argument in another message. For example, the object that is the result of the message - 1 being sent to height is used in each of these ways in moveTower:from:to:using:. Except for the assignment operator (covered in the next chapter), all Smalltalk code is a grand concoction of messages sent to objects.

DEFINING A METHOD

> For *those that like this sort of thing,*
> *this is the sort of thing they like.*
>
> MAX BEERBOHM

The best way to read this section is while sitting in front of a Smalltalk-80 system. (But we supply lots of pictures to help readers who have only an imaginary Smalltalk machine.) Notice that Smalltalk relies on a bit-mapped graphic display and pointing device, rather than a conventional character-oriented display. This difference means little to the Smalltalk language, but a large screen makes programming much easier than on a traditional display terminal. Seeing more than one piece of code at a time relieves a large mental burden.

It will probably save you time if you get a friend to show you how to start Smalltalk, move the mouse, use the mouse buttons, enter windows, select text, bring up menus, and scroll in windows. If you are the first on your block to use a bit-mapped display with a mouse, you can use the system from the hints we give you here, or look in the User's Guide.* We will give you a very careful step-by-step explanation of how to navigate and enter programs. At the back of this book you will find a user interface pocket reference card. *Pull it out and use it as a reminder.*

To start Smalltalk you probably have to do something sophisticated like typing @ST80. If the incantation at your site is different, write it here:

If that doesn't work, ask a friend or use the on-line help facility of the operating system from which you are trying to launch Smalltalk. The display should now look similar to the figure below. It may differ if some windows are closed, collapsed, or not there at all (see Figure 2.1).

We would like to say that there is just one standard way to drive a Smalltalk-80 system, and that this book teaches it. Unfortunately, there are two ways that your system might be different from the standard version, depending on the numbers of mouse buttons** and the

• As mentioned in the preface, "User's Guide" refers to *Smatt.talk-80: The Interactive Programming Environment,* by Adele Goldberg. In the User's Guide, read Sections 1.1 for the mouse, 2.5 for entering windows, 3.1 for selecting text, and 2.3 for scrolling.

** *How many buttons are there on your mouse or pointing device?* If there are three buttons, then the descriptions we will give are correct, and you can skip to the second part of this footnote. We describe the three buttons as left, middle, and right. If your mouse has only one

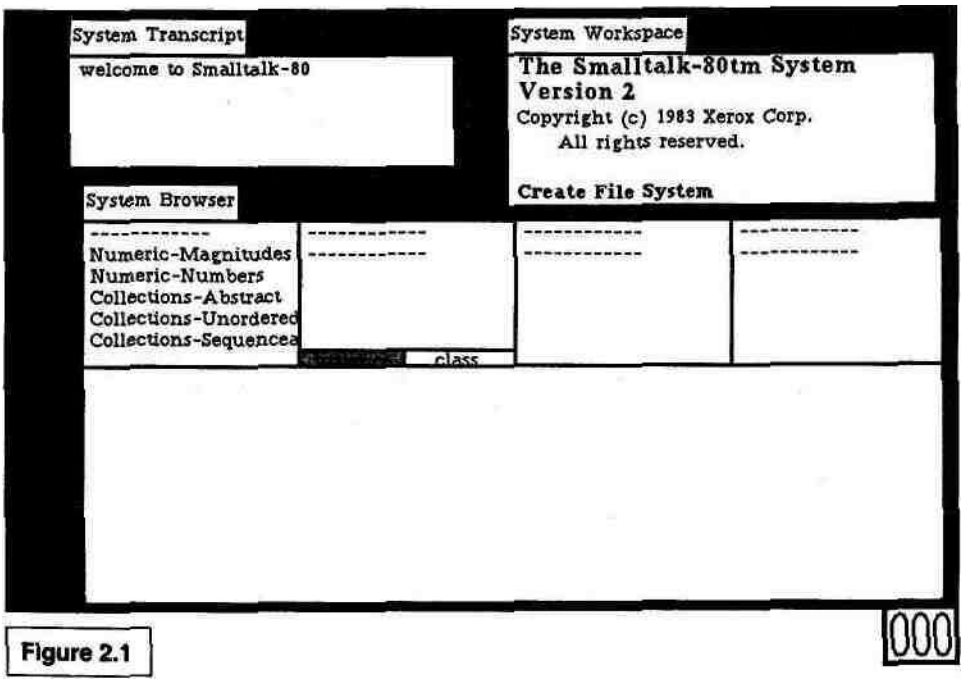

Figure 2.1

version of the Smalltalk system***. We call the three mouse buttons "left," "middle," and "right." In some Smalltalk methods, and in the other Smalltalk books, the buttons are called, respectively, "red,"

button, you will need to do something different when we say "click the middle button." Apple computers use position on the screen in conjunction with their single mouse button to provide three virtual buttons for Smalltalk.

To use the "left button," point at anything "inside" the main part of a window and press the single mouse button.

To use the "middle button," move the cursor into the bar at the left edge of the window. When the cursor changes into a picture of a menu 1-;;, press the mouse button. (The cursor v"4 can change into four different shapes inside the bar, depending on its horizontal position.)

To use the "right button," move the cursor into the title tab at the top of the window and press the mouse button.

Alternatively, if you don't like that way of indicating middle and right buttons, there is another way to do it on Apple machines. Instead of using the position of the cursor to encode the buttons, you can use the "Enter" and "Option" keys as you would use shift keys.

To get the "middle button," press the mouse button while you are holding down the "Option" key.

To get the "right button," press the mouse button while you are holding down the "Enter" key.

*** *Is your Smalltalk-80 system a License 1 system or a License 2 system?* Unless you are using a Macintosh, the answer is probably "License 2" (see the table in the preface to see which license you have). The Smalltalk books by Goldberg and Robson describe the License 2 system, and this book also uses the License 2 conventions. The main difference is the choice of items the user has in the middle-button menu in the browser. If you have License 2, just follow the description in the text. Where possible we will note the License 1 deviations in footnotes or parentheses. Macintosh users may find this slightly inconvenient, but bear with us. (Besides Apple, Digital Equipment and Hewlett-Packard also have the option to sell Smalltalk-80 systems under License 1.)

"yellow," and "blue." These kinds of mix-ups may seem childish, but we are here to explore a young subject.

We start by dealing with the display. The largest window on the screen is labeled System Browser. It is used for browsing through the many snippets of program that make up the Smalltalk system. The browser shows pieces of code according to a classification scheme, and is the main place where code is composed and edited. Looking at the browser, you see menus in four areas across the top. These areas are labeled A, B, C, and D in Figure 2.2. Area E is the "text" section

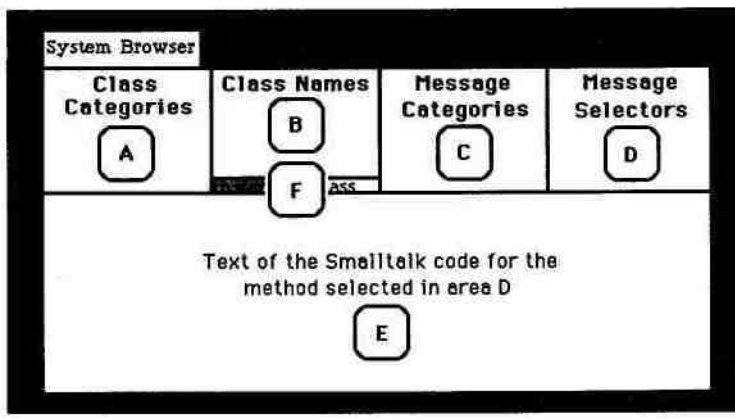

Figure 2.2

where you will edit programs. The broadest categories are in the left menu; after you have chosen one of these categories, a more specific menu appears to its right. You will work from left to right across the four menus, and then see or create a piece of program in the bottom window. For now, don't worry about the significance or meaning of the four menus in the browser.

With traditional programming systems one creates a new program that is loosely linked to other programs via the operating system. In Smalltalk, on the other hand, every program is just a piece of the whole system, and the pieces are linked together. This Zen-like approach to programming means we must find a place for Tower of Hanoi before we can write the program. Let's create a **games** section in one of the more generic parts of the system.

To place the **games** category in the Smalltalk hierarchy:

(1) "Enter the window" of the browser. Do this by moving the cursor inside the browser. One of the windows on your screen is actively listening to your mouse and keyboard. If the label that says System Browser is shown in reverse video (white let-

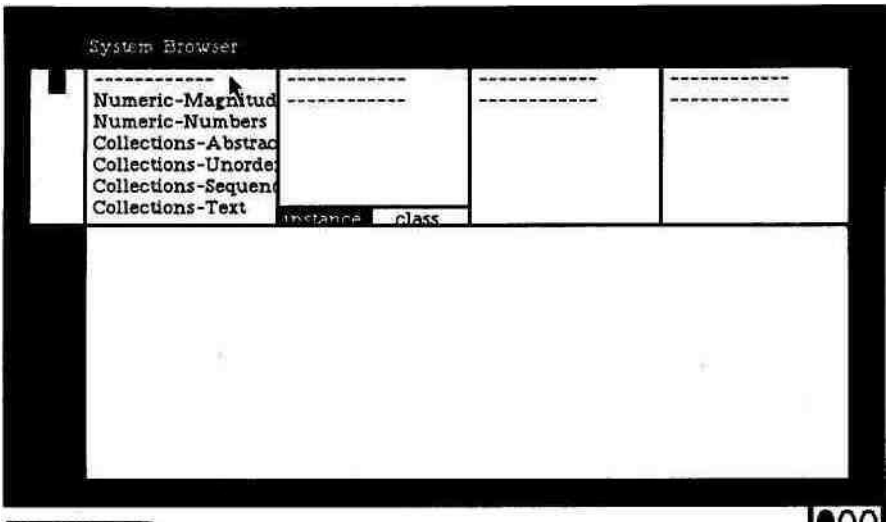

Figure 2.3

ters on a black background), then the browser is the active window. If the browser is not already active, briefly press and release the left mouse button.* This action is called a "click." The System Browser should look like Figure 2.3.

(2) Move the cursor into area A of the browser. Look for **Kernel-Objects,** one of the categories in area A. It is not in the visible portion of area A, so you will have to move the menu to find it. We call this "scrolling." Just move the cursor into the vertical rectangle at the left edge of the window. You will notice the cursor changing shape as you move the mouse from side to side. (If the scroll bar suddenly disappears, move the cursor back into area A.) The horizontal position of the cursor determines which cursor is showing. Move to the right side of the scroll bar until the cursor shows an upward-pointing arrow. When you click, the line beside the cursor will go to the top of the window (see Figure 2.4).

To move the menu down, move the cursor to the left edge of the bar and get the down-pointing arrow. A click now will send the line at the top of the window to where the cursor is (the farther down the cursor is, the more text will scroll down). Try it a few times. You are looking for **Kernel-Objects.** It is near the middle of the list, between the **Graphics-** items

* If you are using a Macintosh 512K system, you are running Apple's "Level 0" Smalltalk-80 system. The browser window may not be on the screen. Instead it is "collapsed" and shows as just a label. If so, put the cursor in the label, press and hold the mouse button, and a pop-up menu will appear. Move the cursor to **frame** and let up on the button. The browser will expand into a window, and you can do Step 1 above.

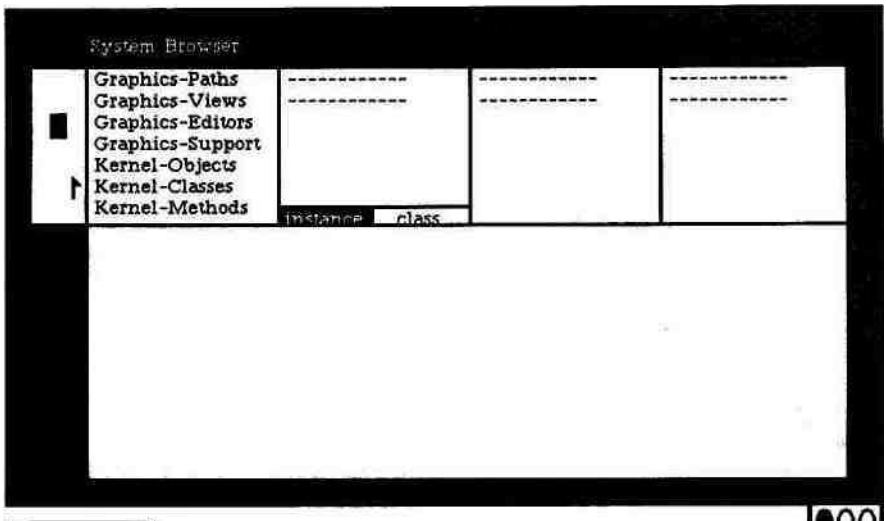

Figure 2.4

and the **Interface-** items (License 1 users will find it after the **Interface-** items). The Smalltalk system allows you to scroll almost any window (see Figure 2.5). If you want to know more about scrolling, see Section 2.3 of the User's Guide.

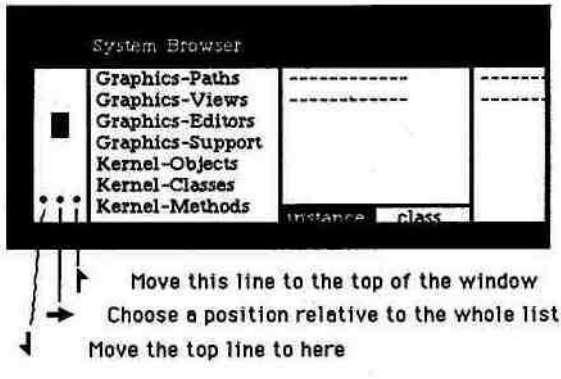

Figure 2.5

(3) Select **Kernel-Objects.** When we say "select" an item we mean place the cursor over the item and click the left mouse button. (Remember? Briefly press and release. The pocket reference card at the back of this book summarizes the various ways the mouse can be used to select an item or move around in the browser. Pull out the card and use it as you read.) In Figure 2.6, the three ovals at the bottom right of the figures represent the left, middle, and right mouse buttons. Any oval that is black means that that button is being pressed. The figure shows the screen as it is before the blackened button is

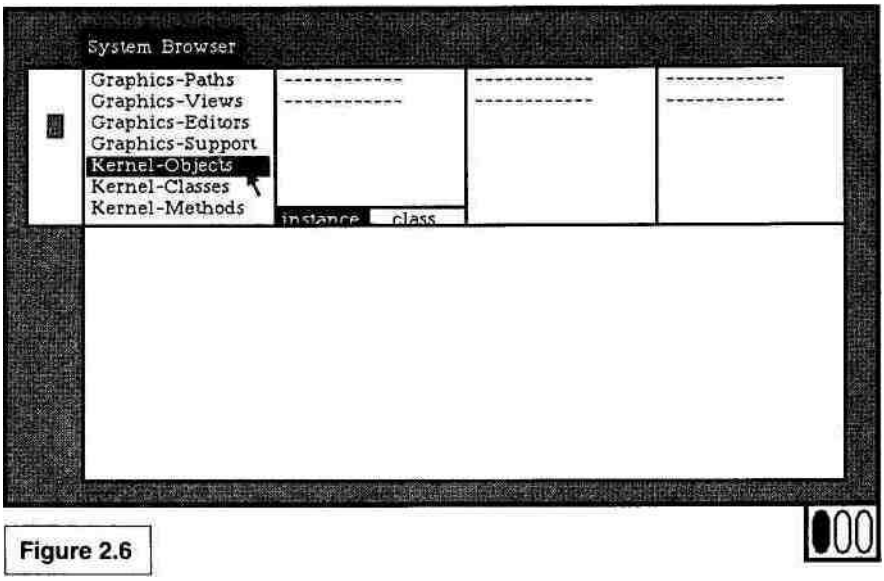

Figure 2.6

released. A new menu will appear in area B after you click on **Kernel-Objects.**

(4) Move to area B and select Object (see Figure 2.7).

(5) In area F, make sure the word **instance** is selected (shown in reverse video). If it is not, click it once. (In this book we will never use the **class** setting, so be sure that **instance** stays selected.)

(6) Now let's add a category in which to put new procedures. Move to area C. Press and hold down the middle mouse but-

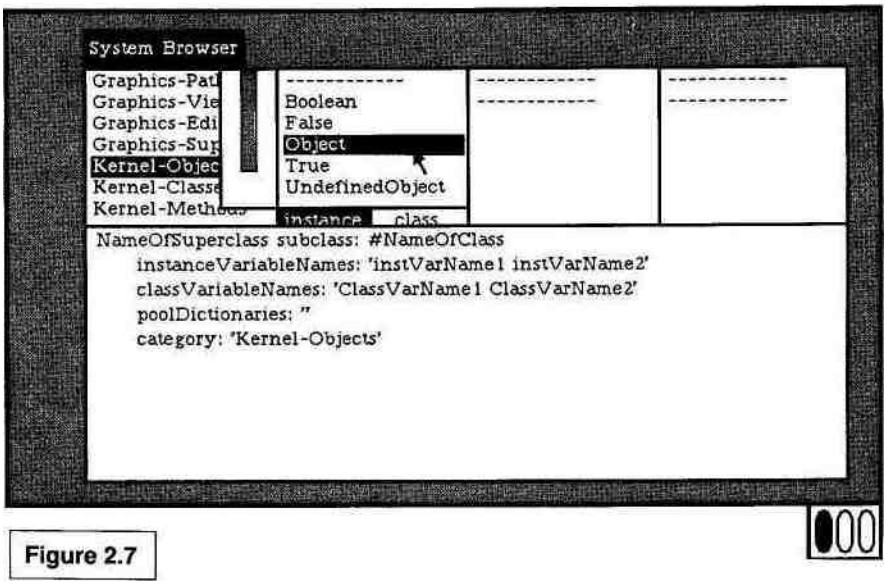

Figure 2.7

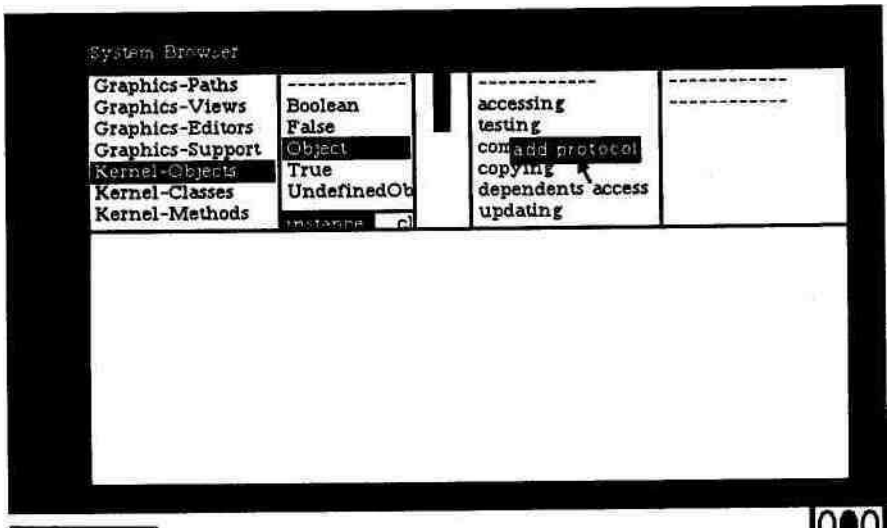

Figure 2.8

ton. (Macintosh users hold down "Option" while pressing the mouse button.) A menu will pop up onto the screen. For obscure technical reasons, this kind of menu is known as a "pop-up menu." This menu has only one item, **add protocol.*** Make sure the cursor is on **add protocol** (it shows in reverse video) and gently release the button to choose the item (see Figure 2.8).

A little window will appear, asking you to type a name. There will be an old name there in reverse video, so just type the word *games* and press the return key (see Figure 2.9). The window will disappear. If nothing happened when you typed, make sure the cursor is inside the little window.

(7) The name of our new section, **games,** will appear and be selected for us in area C. Move the cursor into area E at the bottom of the browser (see Figure 2.10).

You are about to type in the Smalltalk version of the Tower of Hanoi. But first you need to leam about the Smalltalk text editor. Most editors have modes. This means that if you are editing text and you stop to answer the phone, when you return you will have to remember

* If your menu does not have the item • dd protocol on it, you have a License 1 system and must do a little more work. Do not choose any items from the menu (move out of the menu and release the button). Move into area B and hold down the middle button. (Hold down "Option" and the mouse button.) Move to the item called **protocols** and release the mouse button. After a moment, a long list will appear in area E. Move the cursor there, and without clicking anywhere, type ('games') and press return. Be sure the parentheses and single quotes are there. Keep the cursor in area E, and choose **accept** from the middle-button menu (use the "Option" key). Now go on to Step 7.

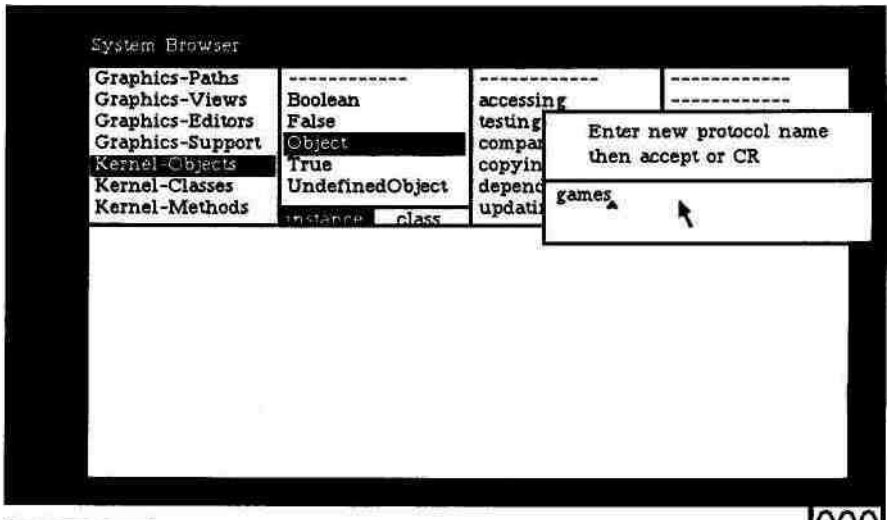

Figure 2.9

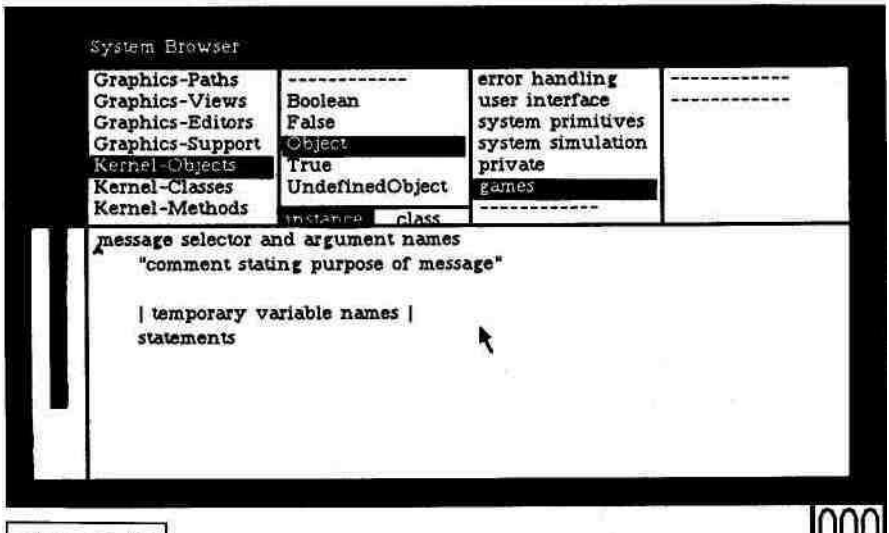

Figure 2.10

what mode you were in (insertion, deletion, or searching) before you can continue. Smalltalk uses a "modeless" editor, meaning that there is nothing to remember. There are fewer commands than in a modal editor, and any of these actions may be performed at any time.

You modify text by clicking somewhere in the text and then typing; what you type goes where you clicked. When you click in the text, a little mark, like this A, appears to tell you where the new text will be inserted.

You select existing text by pointing in front of the first character, holding down the left button, moving the cursor to after the last character, and letting up on the button. Text that is selected is shown in reverse video. As you can see, m^fflm^JJ^^Q, but this text is not.

The middle button gives you a pop-up menu with the standard editor functions (Macintosh users should hold down the "Option" key to get the middle-button menu). You only need a few of these functions now (a full list appears in Appendix 1). To pick one, just hold the middle button, point at the item you want, and release the middle button. This is called "choosing" a menu item. If you are in the middle-button menu and decide not to take any of the choices, move out of the menu and let up on the button.

copy Copies the selected text without removing it (the copy is held in a buffer *off* the screen).

cut Removes what you selected.

paste Replaces the current selection with the last thing you cut, copied, or typed. If nothing is selected to be replaced, it inserts the text where the little mark is.

You replace text by selecting it and then typing. As soon as you type the first character, the old text is cut out. If you have trouble editing and no one is around to ask, consult Section 3.3 in the User's Guide. Before going on, practice by making a bunch of changes in the text in area E (see Figure 2.11). Now that you have had a quick introduction to the editor, we are ready to continue with the example.

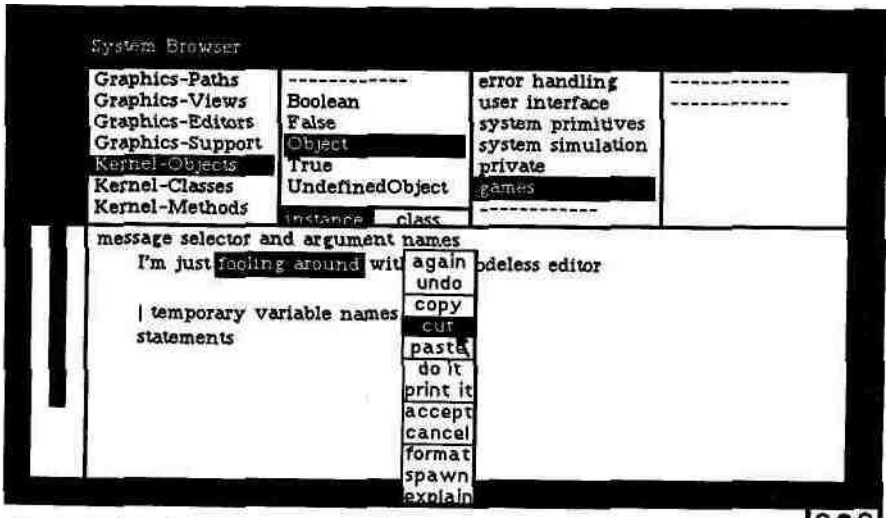

Figure 2.11

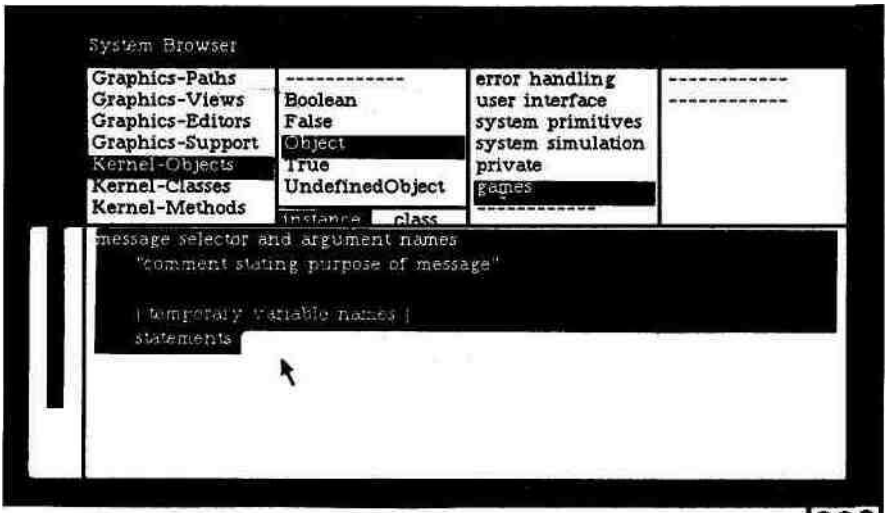

Figure 2.12

(8) In area E, select all the text: Move the cursor to just before the first character and press and hold the left mouse button. Still holding the button, move the mouse down below all the text. Release the button. All the text should be selected; if not, click once somewhere in area E and try again (see Figure 2.12).

(9) What you are about to type will replace the selected text. (You may want to use the tab key to get the proper indentation.) Type the program shown here:

```
moveTower: height from: fromPin to: toPin using: usingPin
  "Recursive procedure to move the disk at a height from one
  pin to another pin using a third pin"
  (height > 0) ifTrue: [
    self moveTower: (height-1) from: fromPin to: usingPin using: toPin.
    self moveDisk: fromPin to: toPin.
    self moveTower: (height-1) from: usingPin to: toPin using: fromPin]

  "This comment gives an example of how to run this program.
  Select the following and choose 'do it' from the middle-button menu.
    (Object new) moveTower: 3 from: 1 to: 3 using: 2"
```

(10) Read over what you have typed. Including the comments, there should be 19 colons, 4 periods, and 4 double quote marks, and possibly some words as well. Do the parentheses and brackets match? Is everything spelled right? Also, remember that capital letters are used as visual separators, so be sure that you have typed the program in exactly as it appears above. To correct something, select the incorrect characters (press on the left button before the first character and let up after the last character), then type the right characters.

(11) Hold down the middle button, move to the **accept** item, and release (that is, "choose" **accept** from the middle-button pop-up menu). See Figure 2.13.

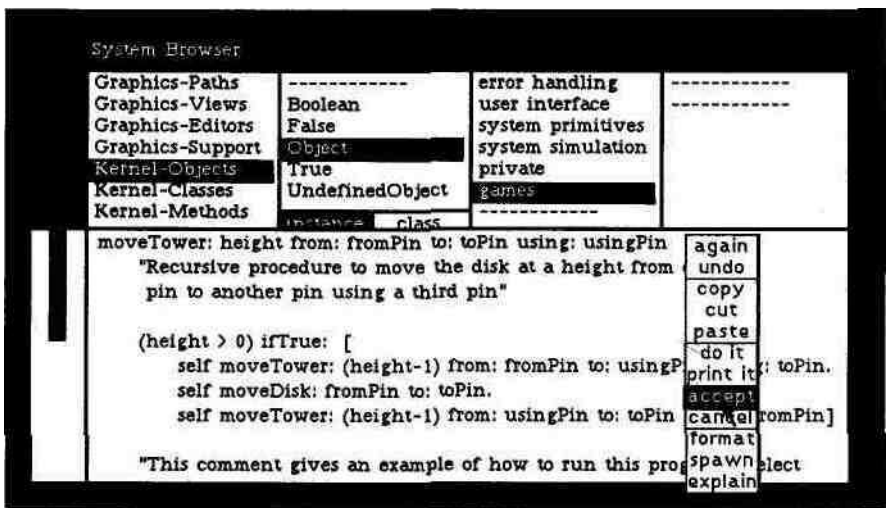

Figure 2.13

This attempts to compile, link, and load what we have just typed. **accept** reads everything in area E, so you don't need to select any text. However, the system will ask you a question before it absorbs what you typed (see Figure 2.14).

(12) This question will appear as a new kind of menu on the screen. When the compiler finds a syntax error, an undefined variable, or a new procedure name, it puts up this menu. Look at what the menu says across the top. It should say, "moveDisk:to: is a new message" because you have not yet defined that method. The system does not recognize the method name and wants to know if you made a typo or are

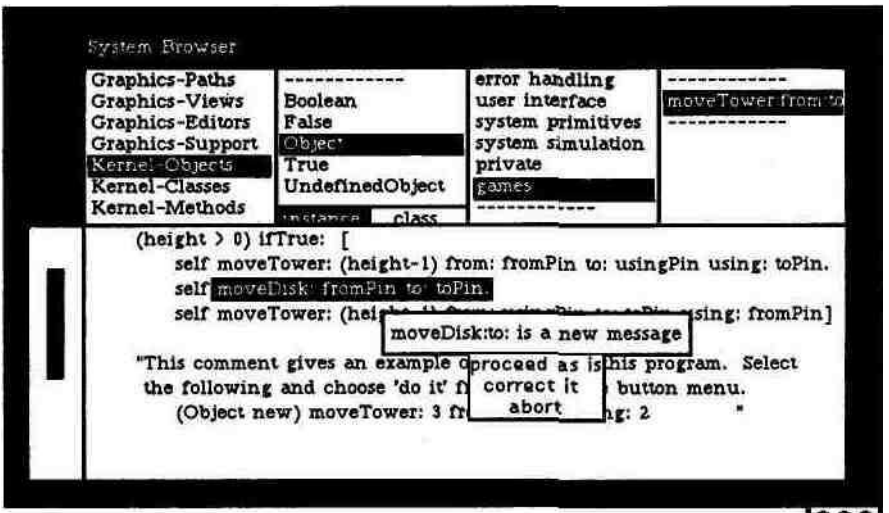

Figure 2.14

just mentioning a new procedure. Reassure the system by clicking **proceed as is.** (In License 1 systems, the menu says **Unknown selector...** across the top. Click on the first item in the menu, which should be moveDisk:to:.) If the menu says something else across the top, or if a little note is inserted into the text, then you've probably made an error typing in this method. To get help in locating the problem, read the section on troubleshooting at the end of this chapter.

When the system successfully accepts our new procedure, it will list the name of the procedure in area D, and show the name in boldface in area E. Don't try to run the program yet; the code you wrote calls another procedure named moveDisk:to: and we haven't typed that in yet. (If you try, the system will say it doesn't understand moveDisk:to:.) In the next chapter we will define that method, and run our program.

Congratulations. You have just grafted a new procedure into your Smalltalk-80 system.

TROUBLESHOOTiNG WHEN YOU accept A METHOD

Let's review what can happen when you ask the system to **accept** a method (procedure) you just typed. If you have any trouble in later chapters, you can refer to this section to help you find the problem. If this section becomes tedious, just skip it and go on to Chapter 3.

- If the method contains any message selector (procedure name) that has not been mentioned before, Smalltalk will put a menu on the screen. When we defined moveTowerfromAccusing: in the previous section, the selector moveDiskto: had never been defined or used. The system put up a menu with the title **moveDisk:to: is a new message** (the menu is shown in Figure 2.14, in the previous section). Since it was indeed a new message, and since it was spelled the way we wanted it, we clicked **proceed as is,** and Smalltalk finished compiling the method. (In License 1 from Apple the menu would appear as **Unknown selector, please confirm, correct, or abort:.)**
- If the menu says something else across the top, you probably have a typo. The questionable phrase is mentioned and is also selected in the text. In the example shown in Figure 2.15, the phrase "to: usingPin" has been left out, and the system didn't recognize the rest of that name. If something like this happens, choose **abort** from the menu (click on it), correct the text, and choose **accept** again from the middle-button menu.

The *(method name)* is a **new message** menu is symptomatic of several different problems. Besides pinpointing a truly new message, or a piece missing from an old message, it may indicate that a period is missing at the end of a statement. If you mean to type:

```
self moveTower: (height -1) from: fromPin to: usingPin using: toPin.
self moveDisk: fromPin to: toPin.
```

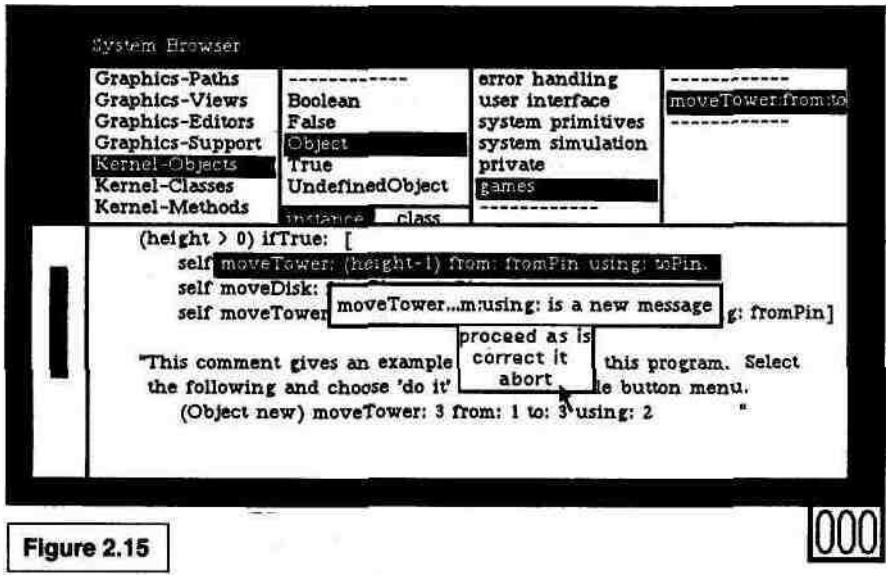

Figure 2.15

but forget the period separating the statements, the compiler will think that the whole thing is one giant message and will ask you about the new message moveTower:from:to:using:moveDisk:to:. If you spell a message name incorrectly, or fail to capitalize the right letters in a message name, the same land of menu will appear.

- If the compiler detects a variable that has never been mentioned before, it puts up a menu that says **declare** *(variable name)* ***as.*** (In License 1 systems from Apple, the menu is titled **Unknown variable:** *(variable name)* ***please correct, or abort:.)*** In the menu are the choices for the type of the new variable. If the variable name is simply spelled wrong, you can invoke the spelling corrector by choosing **correct it** (or, if it shows in the menu, you can choose the correct spelling).
- There is another class of errors which is particularly hard to diagnose. If you type the method in area E of the browser, but previously forgot to choose the right settings in the other areas of the browser, strange things will happen. You will look at your code in area E, and see that it is typed perfectly, but the system will still refuse to **accept** it. If you have failed to select an item in areas C, B, or A, the browser will do one of three things when you choose **accept.** Area E may flash once, the message Nothing more expected: - > may be inserted at the start of the first line, or the system may ignore the **accept** command entirely. If you discover that nothing is selected in area C, select all the text you typed and choose **copy** from the middle-button menu. Then go to areas A, B, and C of the browser, choose the correct settings for this method, and **paste** your method into area E.
- In later chapters we will define several different classes of objects and define methods in each. If you have selected one class in area B (and something in area C), and if you try to **accept** a method that does not belong in this class, you will keep getting menus that ask you to declare variables. The compiler won't recognize the variables that belong to another class. When you realize that the method you typed really belongs in another class, **copy** it from area E, choose the correct class and category in areas B and C, and paste your method into the new area E.
- When the compiler detects a syntax error other than an unknown variable or message name, it inserts a little note into the text such as Argument expected: - >. If this happens, look at the text just after the note and try to find the problem. When you know what needs fixing, **cut** the little note from the text, make the

correction, and choose **accept** again. Error messages like this, which are inserted into the method and highlighted, are simple syntax errors. You will see them when you have unmatched single quotes, double quotes, brackets, or parentheses, or when a message that takes an argument is not followed by one.

For more details on errors that are detected when you choose **accept,** see Chapters 16 and 17 of the User's Guide.