

APPENDIX 1

THE SMALLTALK TEXT EDITOR

To replace a passage of text, select it by pressing the left button at the beginning of the passage and releasing it at the end. Then type the new passage. The first keystroke will delete the old passage.

The middle-button pop-up menu contains the commands used to edit text. This menu is available wherever you can type text.

- again** do the last **paste** again, but in a new place. Find the next occurrence of the text that was pasted over last time. Replace that text.
- undo** undo the last editing action (only works one command back and only if the selection has not moved).
- copy** remember the text that is currently selected.
- cut** remove the text that is currently selected.
- paste** replace the selection with what was last cut, copied, or typed.*
- do it** treat the current selection as Smalltalk code and evaluate it.
- print it** treat the current selection as Smalltalk code, run it, and insert the result after the selection.

* Macintosh users should note that paste will not paste in the last thing typed. It must have been cut or copied. In this respect, the text editor in Apple's version of Smalltalk has been modified to be like the Macintosh text editor.

- accept** compile, link, and load the method (or class definition) in this window.
- cancel** redisplay the text as it was at the time of the last **accept** (undoes all edits since the last **accept**).
- format** pretty print the text for this method; in other words indent the program so it is easy to read. If you like the new form, choose **accept** afterwards. Does not work if you have changed the text since the last **accept**.
- Spawn** creates a new browser, just for this method.
- explain** inserts an explanation of the single thing that is selected. It has trouble if more than one "thing" is selected.

For more detail on the text editor, see Chapter 3 of the User's Guide.

APPENDIX 2

HOW TO TALK TO YOURSELF WHEN READING' SMALLTALK

As we mentioned above, some people feel the need to pronounce when writing programs. We have provided a Smalltalkese reading of `moveTower:from:to:using:` and `moveDisk:to:`.

```
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`

The method for `move-tower-from-to-using`. The arguments are `height`, `from-pin`, `to-pin`, and `using-pin`. (A recursive procedure to move the disk at a height from one pin to another pin using a third pin.) Height is greater than zero, if true, send yourself `move-tower` with height minus one, from `from-pin`, to `using-pin`, using `to-pin`. Send yourself `move-disk` from `from-pin` to `to-pin`. Send yourself `move-tower` with height minus one, from `using-pin`, to `to-pin`, using `from-pin`. Return self ("return self" is the "amen" of Smalltalk). This benediction is implicitly at the end of every method.

moveDisk: fromPin to: toPin

"Move a disk from a pin to another pin. Print the results in the transcript window"

Transcript cr.

Transcript show: (fromPin printString, ' -> ', toPin printString).

The method for move-disk-to. The arguments are from-pin and to-pin. (Move a disk from a pin to another pin. Print the results in the transcript window.) Transcript carriage return. Transcript show from-pin's print string, concatenated with the string for a little arrow, concatenated with to-pin's print string. (This program is not actually doing anything about moving the disks!) Return self (Amen).

APPENDIX 3

METHODS MISSING FROM THE APPLE LEVEL 0 IMAGE

Early versions of the Level 0 Smalltalk system for the Macintosh 512K have some methods missing. The Level 0 system is a cut-down version of Apple's Level 1 system (which is for machines with a megabyte of memory or more). A few classes and many messages were removed to make a small system. The programs in this book happen to use two methods that were taken out, as well as one that was changed. Please follow the directions below to install the missing methods, and then return to denning the method `hanoi` in Chapter 3.

- (1) Enter area A of the browser and scroll to the category **Interface-Browser**. It is above **Kernel-Objects** and is the fourth **Interface-** category. Select **Interface-Browser** by clicking on it.
- (2) Select `MessageCategoryListView` in area B. (Area B may not be wide enough to see all of the name. Of the two names that begin `MessageCategoryLi...`, select the second one.)
- (3) In area C, the system automatically selects **As yet unclassified**. Choose `list:` in area D.
- (4) In area E, all you need to do is add the word `self` and a space to the beginning of the last line. The change is underlined below.

```
list: anArray
    "Refer to the comment in ListView|list:"

    super list: anArray.
    (anArray ~= nil and: [anArray size = 1]) ifTrue:
        [Selection <- 1.
         self controller preSelectModeSelection: 1]
```

- (5) Choose **accept** from the middle-button menu. (You may be wondering what you just fixed. Notice that in Step 3 above, the single item in area C was selected automatically. The bug we just fixed was introduced when that feature was added. When we create a new browser window, as is done in Chapter 4, this code tries to select the only item in area C before the variable controller is initialized. Sending the message self controller instead gets us the same variable, but the code happens to check if it is uninitialized. But wait, we don't yet know enough about Smalltalk to make sense of this.)
- (6) Enter area A of the browser and select the category **Interface-Menus**. It is the category above **Interface-Browser**, the one we were just in. Select **Interface-Menus** by clicking on it.
- (7) Select FillInTheBlank in area B.
- (8) Earlier we said that we would never use the **class** switch in area F of the browser (below area B). Well, now we have to use it just for a moment, and then we will switch it back to **Instance**. Move the cursor down from area B to area F and click on class.
- (9) In area C, the system automatically selects **As yet unclassified**. Look in area D to see if the method request: is there. If it is, you don't have to type it in after all, and can go directly to step 13. Otherwise . . .
- (10) In area E, select all the text and replace it with

```
request: messageString
```

```
"Create an instance of FillInTheBlank whose question is mes-
sageString. Display it centered around the cursor. Return the
string that the user types and accepts."
```

```
self
```

```
request: messageString
```

```
displayAt: Sensor cursorPoint
```

```
centered: true
```

```
action: [response | response]
```

```
initialAnswer: ". "< - two single quotes"
```

```
f response
```

- (11) In the line before the last line, initialAnswer: ". " has two single-quote characters after the colon. Two single quotes in a row is a null String. It is the same thing as (String new: 0). (We also can't resist telling you what this code does. self is the object FillInTheBlank, which is a "class." We will learn about classes in Chapter 4. self is sent the message

request:displayAt:centered:action:initialAnswer:. Because of the block, the local variable response is set as a side effect. In the last line, the method returns the value in the variable response to the caller. We will discuss return in detail later.)

- (12) Choose **accept** from the middle-button menu.
- (13) Move to area F and click on **instance**. *Be sure to do this!* If you leave the switch on **class**, you won't be able to Bnd things in the browser. Now let's define the other missing method.
- (14) Enter area A and scroll to the category **Collections-Text**. It is above the **Interface-** categories and is the fourth **Collections-**category. Select **Collections-Text**.
- (15) Select String in area B.
- (16) In area C, the system automatically selects **As yet unclassified**. If asNumber is already in area D, you can skip to Step 19.
- (17) In area E, select all the text and replace it with

```
asNumber
```

```
"self is a string with the ASCII characters for some digits.  
Convert the digits to a number and return it."
```

```
f Number readFrom: (ReadStream on: self)
```

- (18) Choose **accept** from the middle-button menu. (Both the Apple Level 0 and Level 1 systems are Xerox License 1 systems. If you have a License 2 system and are reading this section anyway, we must tell you that License 2 has a new name for the message on:. In the code for asNumber, you will find ReadStream onCoilection: self instead of ReadStream on: self.)
- (19) Scroll back to **Kernel-Objects**. It is below all the **Interface-**categories. Select **Kernel-Objects** in area A, Object in area B, **games** in area C, and continue with the example on page 38 of the text.

APPENDIX 4

EXERCISES

*No more training do you require.
Already know you that which you need.*
YODA in *The Empire Strikes Back*

To get more experience, modify the animated Tower of Hanoi program to add some bells and whistles. Here are a few suggestions. Appendix 5 contains hints to help you, and Appendix 6 gives some example solutions.

(1) The disks in the animated example are black. Change them to gray.

(2) The disks move from one stack to another by moving directly from their old positions to their new places. Change this so that a disk jumps up above its original stack, jumps across to the new stack, and then jumps down to its final position.

(3) Make the animation pause when any mouse button is pressed.

(4) If you try to use more than 7 disks, the largest ones will overlap each other when they are on adjacent poles. Make the width of a disk depend on the number of disks, so the widest disk is always 80 screen dots wide. Similarly, make the height of the disk depend on the number of disks, so that a full stack of disks is as high as the white rectangle on the screen.

(5) When the game is running and the user presses a mouse button, print (in the transcript) which disks are on each of the three poles.

(6a) Use a `Form` instead of a `Rectangle` for the shape of a disk in class `HanoiDisk`. Color the disk gray and give it a black border that is two screen dots wide. Class `Form` is in the category **Graphics-Display Objects**.

(6b) Use the followrwhile: message in class Form to give the disks smooth movement on the screen. The result should be nice-looking disks and smooth animated movement. Make the disks go in straight lines between their locations, or up and over, or in parabolas.

(7) There is a bug in classes HanoiDisk and HanoiDiskRules. If you create two instances of the game, there will be a conflict in setting the value ofTheTowers. TheTowers is shared by all instances ofHanoiDisk, when it should only be shared by all instances in a single game. Fix this by giving HanoiDisk a new instance variable that performs the same function as TheTowers. Ifyou have done Problem 4, orjust in case you will do it later. Thickness will no longer be a constant, and has the same problem. For completeness, turn every class variable (in class HanoiDisk) that is not a constant into an instance variable.

APPENDIX 5

HINTS FOR THE EXERCISES

The answers can be found on the following pages, but don't peek until you have tried using these hints.

(1) The act of drawing the rectangle is controlled in the method `invert` in class `HanoiDisk`. The code says:

```
invert
  Display reverse: rectangle
```

The variable `rectangle` is a simple `Rectangle` and does not actually have screen bits stored inside it. `BitBit`, Smalltalk's universal bit-slinging algorithm, performs several different types of operations (rules), and each goes through a mask to decide what bits to operate on. Base your changes to `invert` on the definition of `reverse`. Find it by using the **messages** command in the middle-button menu of area D of the browser. (Find the code for `invert` in the browser, then move to area D and hold down the middle button.) The current mask is `Form black`, which means the whole rectangle. `Form gray` is also available.

(2) Modify the method `moveUpon`: in class `HanoiDisk`. The two new stopping points are (rectangle center x @ 120) and (destination center x @ 120). Split the delay up into three equal parts, one for each place the disk shows on the screen.

(3) You can read the mouse buttons by sending messages to `Sensor`, an instance of class `InputSensor` which is found in the category **System-Support**. Adding `Sensor wartNoButton` to the program will cause it to pause unless (or until) all buttons are up. You might want to look at the other messages in `InputSensor` to see what else you can do with the mouse.

(4) The width of a disk is controlled by the constant `14` in the next to last line of the method `width:pole:` in class `HanoiDisk`. Create a new class variable to hold the width increment, and compute the proper

value for it in whichTowers:. When the program runs with N disks, the largest disk has a width of N times the increment and the smallest is 1 times the increment wide. To make the height of a disk depend on the number of disks, make Thickness in whichTowers: be a function of the number of disks.

(5) As in Problem 3, add a line to moveUpon: in class HanoiDisk. The expression Sensor anyButtonPressed returns true if the user is holding a button down. The object that represents the whole game (TheTowers, an instance of AnimatedTowerOfHanoi) should be given the task of reporting the stacks, because an individual disk in the process of moving itself does not know what disks are on other poles. DeBne a new message in AnimatedTowerOfHanoi that prints the report in the Transcript.

(6a) A Form is a rectangle of bits that can be pasted on the screen. It knows its own extent (size), but not its location, so we still need the variable rectangle. Add an instance variable so that each disk can hold a Form. Create a Form by saying

Form extent: rectangle extent.

You can use the message fill:rule:mask: to paint bits into a Form. Look in the classes from which Form inherits its behavior to find the message displayOn:at:clippingBox:rule:mask:, and use it for displaying a Form on the screen.

(6b) The first argument to the message follow:while: should be a block of unevaluated code. It must deliver the next point where the upper left corner of the Form should be displayed. The second argument is another block that returns true until the disk reaches its destination. follow:while: assumes that the image of the disk is not on the screen when it starts to move it, and it does not leave the image on the screen at the end (so we have to compensate).

(7) After you have added an instance variable to the definition of HanoiDisk, you need to find all the places where the class variable you are replacing is used. An easy way to do this is to choose **class var refs** from the middle-button menu in area B. The system will ask you to frame a window, and it will list all of the methods that use the variable. You can see the code by clicking on the method name in the upper pane. Once you are looking at the code, you can modify it and **accept** it.

APPENDIX 6

ANSWERS TO THE EXERCISES

(1) Change the method for invert in class HanoiDisk to be

invert

"Show a disk on the screen by masking an area and reversing it."

Display fill: rectangle

rule: Form reverse

mask: Form gray.

The rectangle is still merged onto the screen using "exclusive or," but this time not all of the bits are changed. Only where the mask is black are bits on the screen reversed. We could have changed this code inside the reverse: method in class DisplayMedium, but since it is used by many parts of the system, all sorts of things (like highlighting in menus) would suddenly behave differently.

Notice that the modification we have made works for both AnimatedTowerOfHanoi and TowerByRules. The disks used by TowerByRules are instances of class HanoiDiskRules and they inherit the methods for displaying themselves from HanoiDisk.

(2) Change the method for moveUpon: in class HanoiDisk to be

moveUpon: destination

"This disk just moved. Record the new pole and tell the user."

pole ← destination pole.

self invert.

"straight up"

rectangle center: (rectangle center x @ 120).

self invert.

(Delay forMilliseconds: 100) wait.

self invert.

"sideways"

rectangle center: (destination center x @ 120).

self invert.

(Delay forMilliseconds: 100) wait.

self invert.

"straight down to final location"

rectangle center: destination center - (0 @ (Thickness + DiskGap)).

self invert.

(Delay forMilliseconds: 100) wait.

(3) When Sensor is sent the message waitNoButton while a mouse button is pressed, it waits until you let go of the button. Insert this line:

```
Sensor waitNoButton.    "wait if button mouse is being held"
```

between any two statements in moveUpon: in HanoiDisk.

(4) Let's make a new variable to hold the difference in width between successive disks. Call it WidthDelta and make it shared by all instances of class HanoiDisk.

First select HanoiDisk in area B of the browser. From the middle-button menu in area B, choose **definition**. Add the class variable WidthDelta, as shown:

```
Object subclass: #HanoiDisk
  instance VariableNames: 'name width pole rectangle'
  class VariableNames: 'Thickness TheTowers DiskGap WidthDeita'
  poolDictionaries:''
  category: 'Kernel-Objects'
```

When you choose **accept** from the middle-button menu in area E, the system determines that WidthDelta is a new class variable, and adds it.

To use WidthDefcta, replace the number 14 in the next to last line of the method width:pole:.

```
rectangle <- 0@0 extent: (size*WidthDelta) @ Thickness.
```

The only hard part of this solution is deciding what values WidthDelta and Thickness should have. The incremental width is equal to 80 divided by the number of disks. The thickness of a disk is the height of the white rectangle (220) divided by the number of disks, minus the space between disks. **Here** is a completely new version of whichTowers in class HanoiDisk:

```

whichTowers: aTowerOfHanoi
  | number |
  "compute the class-wide constants for disks"
  TheTowers <- aTowerOfHanoi.
  number <- TheTowers howMany.
  WidthDelta <- 80 // number, "the widest disk is 80"
  DiskGap <- 2.
  Thickness <- (220 // number) - DiskGap. "divide the height up evenly"

```

You can add a little class to this solution by not letting the disks be too thick. The purpose of making the height vary with the number of disks is to keep the top of the stack on the screen when there are lots of disks. When there are only three or four disks, the disks are quite thick and they don't look as good. Changing the last line to

```

Thickness ← ((220 // number) - DiskGap) min: 14.
  "divide the height up evenly, but not too big"

```

limits the thickness to a pleasing 14 screen dots.

(5) Add a new line of code at the end of `moveUpon:` in class `HanoiDisk:`

```

Sensor anyButtonPressed ifTrue: [TheTowers report].
  "If the button is pressed, ask the whole game to print its state"

```

It is important to put this line at the end of the method because we want to make our report when the state of the disks on the stacks (from which the report will be generated) agrees with the picture on the screen. We pass the task of actually doing the reporting to `TheTowers` in the form of a new message. Now let's write the code for that new message in class `AnimatedTowerOfHanoi:`

```

report
  "Show in the Transcript a written report of which disks are on
  which towers"
  | aStack |
  1 to: 3 do: [:index |
    aStack *- stacks at: index.
    Transcript cr.
    Transcript show: 'Tower number', index printString.
    aStack isEmpty ifTrue: [Transcript show:' has no disks']
    ifFalse: (
      Transcript show:' has disks'.
      aStack reverseDo: [:disk |
        Transcript nextPut: disk name.
        Transcript space]]].
    Transcript cr.
    Transcript endEntry. "force it to show"

```

(6a) Add an instance variable called `image` to class `HanoiDisk`. Initialize it by adding these lines to the end of `width:pole`:

```
size >= 1000 ifFalse: [ "a normal disk"
  image <- Form extent: rectangle extent, "set its size"
  image fill: image boundingBox
  rule: Form over
  mask: Form gray. "fill in the halftone"
  image borderWidth: 2]. "give it a border 2 dots wide"
```

Use `image` as a pattern and invert the bits on the screen where the pattern has black bits. Change the method for `invert` to be

`invert`

```
"Show this disk on the screen by inverting the bits where the Form is black"
image displayOn: Display
  at: rectangle origin
  clippingBox: Display boundingBox
  rule: Form reverse
  mask: Form black
```

(6b) This solution is for a straight-line path between the disk's starting and ending positions. We start with the code for `moveUpon`: as it appeared before you worked any of the other exercises. All we have to do is to send the message `follow:while:` to the disk's `image`, and insert this between the call on `invert` and the code for moving the rectangle. We also need to define and initialize the local variables that hold the amount to move at each step and the number of steps completed.

```
moveUpon: destination | count endPoint increment |
  "This disk just moved. Record the new pole and tell the user."
  pole <- destination pole.
  "Find the increment to move in a straight line path in 16 small steps"
  count <- 0.
  endPoint <- destination center - (0@(thickness+DiskGap)).
  increment <- endPoint - rectangle center //16.
  "remove the old image"
  self invert.
  "Move along the path. First block is next point, second is end condition."
  image follow: [rectangle moveBy: increment, rectangle origin]
    while: [(count <- count + 1) <= 16].
  "final position"
  rectangle center: endPoint.
  "display at its final position"
  self invert.
```

You can make the disks travel any path you want by varying the code that supplies `Points` to `followWhile`. Try parabolas or semicircles.

(7) Choose class `HanoiDisk` in area B, and use the menu item **definition** to get its definition into area E. Add instance variables `theTowers`, `thickness`, and `widthDelta` (not capitalized to distinguish them from the class variables). As mentioned in the hint, choose **class var refs** to get a little browser on the methods that use each of the class variables. In each method, replace the class variable with its corresponding new instance variable. After accepting each of these changes, we must make sure the new instance variables are assigned values in every `HanoiDisk` that is created. To do this, we need to modify `setUpDisks` in `AnimatedTowerOfHanoi`. Previously, `whichTowers` was called just once in each game to initialize the class variables in `HanoiDisk`. Instead let's call it once for every disk that is created.

```
setUpDisks | disk displayBox |
  "Create the disks and set up the poles."
  "Tell all disks what game they are in and set disk thickness and gap"
  displayBox *- 20@100 corner: 380@320.
  Display white: displayBox.
  Display border: displayBox width: 2.
  "The poles are an array of three stacks. Each stack is an
  OrderedCollection."
  stacks <- (Array new: 3) collect: [:each | OrderedCollection new].
  howMany to: 1 by: -1 do: [:size |
    disk <- HanoiDisk new whichTowers: self. "Create a disk"
    disk width: size pole: 1.
    (stacks at: 1) addFirst: disk. "Push it onto a stack"
    disk invert "show on the screen"]].
```

```
"When a pole has no disk on it, one of these mock disks acts as a bottom
disk. A moving disk will ask a mock disk its width and pole number"
mockDisks <- Array new: 3.
1 to: 3 do: [:index |
  disk <- HanoiDisk new whichTowers: self. "Create a disk"
  mockDisks at: index put: (disk width: 1000 pole: index)].
```

Note that we removed the line in which `whichTowers` used to appear. We also need to make the same modification to `setUpDisks` in `TowerByRules`. (It's not exactly the same modification—we are creating a new instance of `HanoiDiskRules` instead of a new instance of `HanoiDisk`.)

Now you can start one game, interrupt it, and start a second game with a different number of disks. The two games interfere with each other only by occupying the same space on the screen; they no longer try to use the same variables.

INDEX

Entries in sans serif type refer to message (or procedure) names; entries in **Sans serif boldface** refer to menu commands.

abort, 26, 40

abort, 26

abs (absolute value), 69

accept, 58

accept, 24-25, 27-28, 113

active windows, 17

add a category, *see add protocol*

addFirst, 47

addition, 30, 69

of class variables, 125

of instance variables, 127

add protocol, 19, 20, 58

again, 113

aggregate data types, 46

algorithms, 44, 83-98, 110

rule-based, 109

alphabet, 47

"and" (logical), 91

animation, 64-82, 120

anyButtonPressed, 123, 126

area A, 18

area C, 19-20

"protocols" in, 49

area D, 49

area E, 20, 23

area F, 19

Argument expected, 27-28

arguments, 28

to blocks, 87, 90

changes to, 38

input, 83

messages without, 37

in moveTower, 8, 10

in Smalltalk, 11

arithmetic operators, 13, 37, 69

Array, 73-74

arrays, 46, 74

of characters, *see strings*

indexing of, 46-47

subscripts of, *see at*:

arrows:

cursor, 17

down-pointing, 17, 32

left, 36

up, 17, 70, 88

ASCII characters, 47

asNumber, 36, 119

assignment operators, 13, 36

at, 46, 74

at:put:, 73

axes, coordinate, 69

backups, 109

binding of procedure names, *see*

objects, creating of; sending of
messages

BitBlt, 73, 122

bit-mapped graphic displays, 14

black, 120

black:, 73

blocks, 10, 12, 46, 91

arguments to, 87, 90

evaluation of, 90

of unevaluated code, 86

"Blue Book", 59, 112

Boolean expressions, 10, 86, 91

borderwidth:, 73

brackets, 24, 28, 46

curly (Pascal), *see blocks*

square, 10

break, *see control C*

browse, 51

browser, 16, 17, 58, 99

area A, 18

area B, 19

area C, 19-20

browser (*continued*)

- area E, 23
- area F, 19
- class, 101
- creating of, 51, 107
- method, 103, 105
- new**, 51
- spawning of, 63, 106-7
- subclasses in, 85

browser, 107

- bugs, 121
 - see also* troubleshooting
- button down, 123
- buttons, *see* mouse buttons

- calling, 11, 83, 105
 - of procedures, 12, 35
 - of programs, 40
 - of stacks, 35

cancel, 38, 113

- capital letters, 9, 24, 27, 66
- carets, 21

carriage return:

- in input, 39, 91
- in output, 30
- in programming, 37

case of characters, *see* capital letters

case-sensitivity, 24

case statements (Pascal), *see* if-then-else

categories, 16

- adding of, 19, 20, 58
- in classes, 99, 100

center, 70

center:, 70

Change-Management Browser, 109

changes, 109

character-oriented displays, 14, 30

characters:

- arrays of, *see* strings
- ASCII**, 47
- ease of, *see* capital letters
- creation of, 47
- literal, 37
- punctuation, *see* punctuation
- vertical bar, 38, 46

Clancy, Michael, 2

C language, 5, 10, 84, 111

class browser, 101

Class Browser, 51

classes, 27, 44-57, 59, 99

- adding of, 58
- categories of, 49
- changes **in**, 98
- comments for, 45
- defining of, 65, 100

- designing of, 110
- finding of, 100
- hierarchies in, 56
- instance of, 45, 61
- messages in, 108
- as "modules," 99
- sending messages to, 107

Class setting, 19

- class variables, 66, 103, 123, 121
 - adding of, 125
 - converting of, to instance variables, 128
 - finding of, 103, 123
 - new, 125

class var refs, 103, 123

clean codes, 111

- click, 17, 21, 38, 59
 - double, 53

close, 35

closing of windows, 59

closure (LISP), *see* instance variables

code files, format of, 43

codes, 16

- clean, 111
- file in, 42, 60
- file out, 60
- interruption of, 100
- modular, 109
- reading of, 36
- reusing of, 57
- share**, 57
- unevaluated, 86

code talkers, 13

collect:, 47

collection, 46, 59

colons, 11, 37

Command period, 40, 100

commands, 57-63

commas, 30

comment, 10

- for class, 45

comment, 50, 101compilation, *see* **accept**

compilers, 24

compile-time errors, 25

computerese, 57

concatenation, 30

conditional expressions, 68, 87

confirmation, 38

constants, 11

float, 34

- integer, 12

control, flow of, 68, 87

control C, 40, 100

controllers, 81, 89

- control structures, 13, 68, 87, 89-90
- conversion:
 - of class variable to instance variable, 128
 - of digits, *see* `asNumber`
 - of numbers, 46
 - to strings, *see* `printString`
- Cooper, Doug, 2
- coordinate system, 69
- COPY**, 22, 113
- corner cursors, 51
- correct it**, 27
- counter variables, 46
- crash, 109
- cr messages, 30
- curly bracket notation, 10
- cursor, 16-17
 - arrow, 17
 - corner, 51
 - shape, 17
- CÙt, 22, 113

- data, protection of, 109-10
- database:
 - of callers, 105
 - of methods, *see* browser
 - of syntax, *see* **explain**
- data structures, 44
- data types, *see* classes
- debug**, 35, 101
- debugging, 35, 82
 - see also* troubleshooting
- decimal points, 38
- declarations, 111
- default templates, 38
- definition**, 77-78, 125
- deque (Pascal), *see* `OrderedCollection`
- deselection, 31, 38, 60
- design, 110, 112
- DemTs DP Dictionary* (Kelly-Bootle), 1
- diagnose, 25
- dictionary, of Small talk, 93, 108
- digit conversion, *see* `asNumber`
- disks:
 - fake, 68
 - mock, 67-68, 75
 - number of, 120
 - smooth movement of, 121
 - stack of, 2, 44, 46, 120
 - storage, 41
 - width of, 120
 - wooden, 2, 44, 65, 109, 120
- Display, 73, 108
- DisplayMedium, 73, 124
- DisplayObject, 73

- `displayOn:at:clippingBox:rule:mask:`, 123
- Display reverse, 71
- displays, 16, 65
 - bit-mapped, 14
 - character-oriented, 14
 - of Form, 123
 - inversion of bits on, 127
- DisplayScreen, 73
- division, 69, 126
- do:, 46
- do it**, 32, 60, 113
- "do loops," 46
- double click, 53
- double quotes, 10, 28
- down-pointing arrows, 17, 32

- editing, 20, 40, 60
 - of programs, 16
- editors, text, 113
- elements, of stacks, 46
- endEntry, 48
- "Enter," 15
- entering of windows, 60
- equality, 69
- errors:
 - compile-time, 25
 - runtime, 35
 - syntax, 24, 27
 - in typing, 25
- error windows, 33, 35
- escape from execution, *see* control C
- escape from method, *see* carriage return
- eval (LISP), *see* **do it**
- evaluation:
 - of blocks, 46, 90
 - order of, 37
- "exclusive or," 69
- execution stacks, 101
- exercises, 120-21
 - answers to, 124-28
 - hints for, 122-23
- exit, 41-42, 91
- explain**, 101, 114
- expressions, 10, 37, 42, 70
 - Boolean, 10, 86, 91
 - conditional, 68
- extensible languages, 87
- extent:, 123

- fake disks, 68
- false, 10
- false, 68
- file in, 42
- file in codes, 42, 60

file list, 107**file out, 41**

file out codes, 60

files:

reading of, 42

writing of, 41

till:rule:mask:, 123

FillInTheBlank, 36

fill-in-the-blank windows, 36

fixed menus, 18-19, 38, 58, 61, 62

floating point numbers, 34

flow of control, 68, 87

follow:while:, 121, 123, 127

for (Pascal), *see* do:

for-loops, 87

Form, 73, 120-23

format, 114

FORTRAN, 109

frame, 51

framing of windows, 51, 61

function, *see* methods

game-wide information, 75

"gets," 36

global resources, 108

global variables, 66, 93, 108

glossary, 57-63

Goldberg, Adele, 59

graphics, 64-82

Graphics-Primitives, 70

gray, 120

gray:, 73

greater than, 34

greater than or equal to, 75

Grogono, Peter, 3

halt, 100

hanoi, 36, 45

HanoiDisk, 66

hanoi method, 38-41

hierarchies:

in classes, 56, 106-7

of operators, 37

in Smalltalk, 16-20

hierarchy, 85, 107

high-level languages, 110

Horn, B. K. P., 6

hyphens, 9

identification of objects, 36

identifiers, *see* message selectors; variables

ifFalse:ifTrue:, 68

if-statement, 10**if-then-else, 68, 87**

ifTrue:, 68

ifTrue:ifFalse:, 68

increments, 46

indentation, 37**indexing:****of arrays, 46-47, 74**

of stacks, 44

induction, 3

infinite loops, *see* control C

information, game-wide, 75

inheritance, 56, 82, 108

chains, 85

multiple, 108

initial value of variables, *see* nil

input, 36, 122

carriage return in, 39, 91

parameters of, 39-40, 83

InputSensor, 122

inspect messages, 108

inspector windows, 93, 108

instance, 19

instances, 61

of classes, 45, 61

multiple, 121

setting of, 19

instance variables, 66, 93, 89, 108

adding of, 127

converting class variables to, 128

finding of, 101-2

inst var refs, 102

integers, 47, 69

isEmpty, 91

iteration, 46

iteration variables, 46

Kelly-Bootle, Stan, 1

Kernel-Objects, 18

Krasner, Glenn, 43

labels, *see* message selectors; variables

LAMBDA, 87

languages:

extensible, 87

high-level, 110

"production system," 84

see also C language; LISP; Pascal;

Smalltalk

left-arrow, 36

left-arrow key, 38

left-button menus, 38

letters, *see* characters

license 1 Smalltalk, 15, 18, 25, 50

Level 0, 117

- License 2 Smalltalk, 15
- lightGray:, 73
- link, 24
- LISP, 6-7, 34, 87
- list, 117
- lists, 102, 108
 - see also* fixed menus
- literal characters, 37
- literal strings, 30, 37
- load, 24
- local names, 10
- local procedures, 87
- local variables, 36, 46, 66
- logical operators, *see* Boolean expressions
- logout, 41-42
- loops, 46
 - "do," 46
 - for-, 87
 - infinite, *see* control C
 - "main," 85
 - "while," 86
- Macintosh 512K system, 15, 17, 117
- macro, 110
- "main loops," 85
- mask, 122, 124
- measurements, 108
- memory storage, 83
- menus, 14, 16, 22
 - choosing items on, 20, 22
 - fixed, 18, 38, 58, 61, 62
 - left-button, 38
 - middle-button, 22, 38-39, 50
 - pop-up, 20, 38, 50, 59, 62-63, 113
 - right-button, 38
 - right-button pop-up, 35
- Message not understood, 35
- messages, 12, 61, 109
 - without arguments, 37
 - class, 108
 - cr, **30**
 - finding senders of, 105
 - if-then-else, 68, 87
 - multiple implementation, 105
 - names of, 13
 - new, 24
 - order of, 37
 - same, to different objects, 57
 - sending of, 12, 62, 110-11
 - terminating of, 88
 - unfamiliar, 104
 - messages**, 105, 122
- message selectors, 11-12, 26, 61, 101
- method browsers, 103, 105
 - methods, 9-28, 13, 24, 29, 61
 - accepting of, 58
 - adding of, 58
 - callers of, 105
 - categories of, 49
 - changing of, 55
 - creating, of, 111
 - database of, *see* browser
 - defining of, 14-25
 - hanoi, 38-41
 - modifying of, 55
 - moving and, 51
 - overriding of, 68, 92, 100
 - vertical bars in, 36
 - middle-button menus, 22, 38-39, 50
 - min, 126
 - mock disks, 67-68, 75
 - models, 81
 - "modeless" editors, 21
 - modes, of editors, 20
 - modular codes, 109
 - modules, 111
 - mouse buttons, 14-16
 - blue, 15-16
 - left, 15, 17, 18, 22, 38, 53
 - middle**, 15, 22, 38-39, **50**
 - reading, 122
 - red, 15
 - right, 15, 18, 35, 38
 - yellow, 15-16
 - moveDisk-to:, 29-31, 48, 115
 - move methods, 51
 - movetower, 4-5, 11
 - moveTower:from:to:using:, 7, 11, 23, 115
 - multiple inheritance, 108
 - multiple instances, 121
 - multiple screens, 108
 - multiplication, 69
 - nested procedure calls, 35
 - nesting, *see* brackets; expressions; **indentation**; **parentheses**
 - new:, **46**
 - nextPut, 48
 - nil, 48
 - in LISP, *see* false
 - not (Boolean), 91
 - notation, 10-11
 - curly bracket, 10
 - not equal, 90
 - Nothing more expected, 27
 - numbers, 36
 - conversion of, 46
 - floating point, 34

- object-message paradigm, 13
- object-oriented programming, 8, 109-12
- objects, 11, 62, 82, 84, 93, 98, 109, 110
 - creating of, 46
 - describing types of, 45
 - identification of, 36
 - Inspect messages to, 108
 - lists of, 108
 - sending of messages to, 12, 57
 - simulation, 13
- Oh! Pascal* (Cooper and Clancy), 2
- "on the stack," 83
- operands, 12
- operating system, 14, 16, 108
- operators, 11-12, 30
 - arithmetic, 13, 37, 69
 - assignment, 13, 36
- "Option," 15, 20, 22
- "ordered collection," 46, 47-48
- OrderedCollection, 91
- ordinal type (Pascal), *see* Boolean
 - expressions; characters; integers
- output, 32
 - carriage return in, 30
 - see also print it*; Transcript
- overriding of methods, 68, 92, 100

- "package," 99
- parameters, *see* arguments
- parentheses, 24, 28, 37
- partitioning of problems, 110
- Pascal, 3-5, 9-13, 34, 66, 111
 - procedures in, 13
 - records in, 11
- paste**, 22, 113
- path, 123, 127, 128
- pause, 69, 120
- pegs, 2
- periods**, 10, 26-27, 37
- pins, 2
- pocket reference cards, 14, 18
- pointers, *see* objects
- pointing, 38
 - see also* mouse buttons
- points, 69
- poles, 2, 68
- pool variables, 108
- pop-up menus, 20, 38, 50, 59, 62-63, 113
- precedence, 37
- predicates, *see* Boolean expressions
- printing, *see* output
- print it**, 108, 113
- printString, 30, 34

- problems, *see* troubleshooting
- problem-solving, 110
- procedure calls, nested, 35
- procedure names, 11, 12, 26, 61
- procedures, 9, 11, 13, 111
 - calling of, 12, 35
 - definition of, 13
 - exiting from, 91
 - local, 87
- proceed as is**, 25, 26
- Processor, 108
- "production system language," 84
- programming:
 - carriage return in, 37
 - object-oriented, 8, 109-12
 - rule-based, 84, 109
 - style of, 109
- Programming in Pascal* (Grogono), 3
- programs:
 - calling of, 40
 - editing of, 16
 - running** of, 31, 35
 - saving, on disk, 41
 - simulation, 13
 - see also* methods
- "Projects," 108
- property lists** (LISP), *see instance* variables
- protection**:
 - of data, 109-10
 - by subclassing, 82
- "protocols," 49
- punctuation, 9, 10-11, 29-30, 36-38, 101

- queue, 91
- quit, 41-42
- quit**, 42, 109
- quotes:
 - double, 10, 28
 - single**, 28, 30

- "reaching in the back door," 109
- real numbers, 34
- receivers, 12, 62
- records:
 - in Pascal, 11
 - see also* classes
- recover, 109
- Rectangle, 73, 120
- rectangles, 70-71
- recursion, 1-8, 13, 83
- removeFirst, 48
- repeat-until, 87
- replace, 113

- report, 123, 126
- request:, 36-37, 118
- reserved **word**, *see* self; super
- result, 13, 37
- return, *see* carriage return
- return multiple values, 88
- return of value, 12
- reverse:, 122
- revisions, 55
- right-button menus, 38
 - pop-up, 35
- Robson, Dave, 59
- rule-based algorithms, 109
- rule-based programming, 84, 109
- rules, 37
- running of programs, 31, 35
- runtime errors, 35

- save**, 109
- saving, 41, 108-9
- scope, *see* variables
- screens, 71, 120, 123
 - multiple, 108
 - see also displays*
- scroll bar**, 17
- scrolling, 14, 17-18, 62
- searching, 108
 - templates for, 105
- selectors, 12-13, 18, 101
 - current, 22, 113
 - message, 11-12, 26, 61, 101
 - unknown, 25
- select text, 63
- self, 13, 57, 63, 70
- senders**, 105
- sending of messages, 13, 62, 110-11
- Sensor, 122, 125
- separators**, 37-38
- share codes, 57
- shared variables, 66
- Show., 30, 48
- simulation programs, 13
- single quotes, 28, 30
- SmallInteger**, 69
- Smalltalk:
 - arguments in, *see* arguments
 - dictionary, 93, 108
 - exiting from, 41-42
 - global resources in, 108
 - hierarchies in, 16-20
 - Level 0, 17
 - License 1, 15, 18, 25, 50, 117
 - License 2, 15
 - methods in, *see* methods
 - productivity of, 57
 - punctuation in, *see* punctuation
 - recursion in**, 7-8, 13, 83
 - starting of, 14
 - as subroutine library, 99
 - syntax of, 36
 - terminology of, 57-63
 - text editor in, 20
 - type declarations in, 46
 - version 2, 14-15
- Smalltalk-80: Bits of History, Words of Advice* (Krasner), 43
- Smalltalk-80: The Language and its Implementation* (Goldberg and Robson), 59
- Smalltalkese, 13, 57, 115
- snapshots, 41
- spaces, 37
- spawn**, 51, 114
- spawning of browser, 63, 106
- spelling correctors, 27
- square brackets, 10
- stacks, 8, 35, 46, 48, 91
 - calling of, 35
 - of disks, 46
 - elements in, 46
 - execution, 101
 - indexing of, 44
 - see also OrderedCollection*
- statements, 10, 26-27, 37-38, 86
- stop execution, 40, 100
- storage, 83
 - on disk, 41
- strings, 30, 36
 - literal, 30, 37
 - see also* characters
- structure (LISP), *see* classes
- style of work, 108, 111
- subclasses, 67, 82, 92, 100, 106, 108
 - in browser, 85
- subprograms, 110
- subroutine libraries, 99
- subroutines, 9, 13, 111
- subscripts, array, *see at*:
- subtraction, 34, 69
- "vector," 70
- super, 92
- superclasses, 108
- syntax, 10
 - database of, *see explain*
 - Smalltalk, 36
- syntax errors, 24, 27
- System Browser, 7
- System Transcript, 30-31, 107
- System Workspace, 42, 107, 108

- tabs, 37
- templates, 38, 108
 - for searching, 105
- terminals:
 - character-oriented, 30
 - see also* displays
- terminating of messages, 40, 88
- text:
 - click in, *see* click
 - replacing of, 22
 - selection of, 14, 22, 63
- text editors, 20
- then, *see* ifTrue
- tildes, 90
- title tabs, 15
- tokens, 37, 101
- Tower of Hanoi, 83, 109
 - in C, 5, 10, 84, 111
 - defined, 1
 - in LISP, 6-7
 - modifications to, 120-21
 - in Pascal, 3-5, 9-13
 - picture, 2
 - recursion, 1-8, 13, 83
 - in Smalltalk, 7-8
 - subclasses of, 67, 82, 92, 100, 106, 108
- Transcript, 30
- transcripts, 120
- Transcript Window, 8, 31
- transformation of aggregate data types, 47
- troubleshooting**, 25-28
 - of runtime errors, 35
- true, 10
- true, 68
- type declarations, 46
- typing, 22, 113
- typos, 26

- undefined variables, 24
- underscoring, 38-39
- undo**, 113
- unevaluated codes, 86
- Unknown selector**, 25
- Unknown variable**, 27
- until (Pascal), *see* whileFalse
- up-arrows, 17, 70, 88
- User's Guide, 14, 18, 57, 112

- Var (Pascal), 67
- variables**, 11, 45, 101
 - class, 66, 103, 121, 123
 - counter, 46
 - declaring of, 36
 - global, 66, 93, 108
 - instance, 66, 89, 93, 108
 - iteration, 46
 - local, 26, 46, 66
 - new**, 27
 - pool, 108
 - shared, 66
 - undefined, 24
 - unknown, 27
- variant records (Pascal), *see* instance
 - variables; objects
- vectors, *see* arrays
- "vector subtraction," 70
- vertical bars, 36, 38, 46
- views, 81
- visual separators, 24

- waitNoButton, 122, 125
- whileFalse, 86
- "while loops," 86
- white:**, 73
- windows, 20, 29, 81
 - active, 17
 - closed, 14
 - closing of, 59
 - collapsed, 14
 - entering of, 14, 16-17, 60
 - error**, 33, 35
 - fill-in-the-blank, 36
 - framing of, 51, 61
 - inspector, 93, 108
 - refreshing of, 48
 - scrolling of, 14, 17-18, 62
 - System Browser, 7
 - Transcript, 7, 31
 - Winston, P. H., 65
- wooden disks, 2, 44, 65, 109, 120
- workspace**, 107

- X-Y pairs, 69

- Yoda, 120

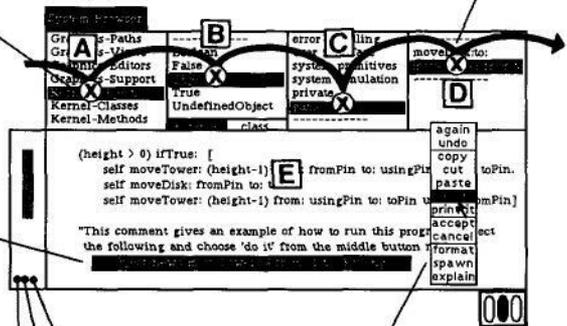
Smalltalk-80™ User Interface License 2 Systems

A window is active when its title shows in reverse video. To use a window that is not active, move the cursor into any part of the window and click (press and release) the left button.

Select an item from a fixed menu by clicking on it with the left button.

Choose items left to right, general to specific.

To select text, press and hold the left button at the beginning of the passage, move to the end, and then release (the selected text will appear in reverse video). New text typed in Area E (the cursor must be in Area E) replaces the selected passage.



Scroll bar Use the left button. The cursor changes shape as you move it from side to side within the scroll bar.

Click to bring the top line of the window down to the same line as the cursor.

Press the button and move up and down in the scroll bar to choose which part of the contents will show. The gray bar represents the fraction of the document that is currently visible.

Click to bring the line beside the cursor to the top of the window.

Pop-up menu Press and hold the middle button, move to the desired menu item, and release to choose it.

Mouse

The left button (red) selects text or fixed-menu items.

The middle button (yellow) controls pop-up menus for editing commands.

The right button (blue) controls pop-up menus for window commands.

A Taste of Smalltalk

Ted Kaehler
Dave Patterson

*Seeing is deceiving.
It's eating that's believing.*
— James Thurber

Written by two Smalltalk experts, this entertaining introduction to Smalltalk-80TM offers a brief tour of both the interactive programming environment and the language, for readers with some programming experience. Step-by-step instructions (accompanied by many pictures of the display screen) help the reader explore the unique user interface, while a series of example programs demonstrates the power of object-oriented programming.

Taking the Tower of Hanoi puzzle as their example, the authors compare a recursive Smalltalk program to similar programs in Pascal, C, and LISP, and then enhance their example with simple animation and a fully object-oriented algorithm. Observations on the nature of Smalltalk and advice on speaking "Smalltalkese" highlight the differences between Smalltalk and conventional programming environments.

A Taste of Smalltalk includes exercises (with hints and answers following) and a tear-out pocket reference card to aid the reader in exploring the system. The manuscript was tested extensively at Xerox's Palo Alto Research Center (PARC), where it was used to let new Smalltalk programmers "get their feet wet," and by students at the University of California at Berkeley.

TM*Smalltalk-80 is a registered trademark of the Xerox Corporation.*

TED KAEHLER was for eleven years a member of the Learning Research Group (later the Systems Concepts Group) at Xerox PARC, after receiving an M.S. in computer science from Carnegie-Mellon University. While at Xerox, he worked on virtual memory problems in Smalltalk (as well as other systems and language concerns), and created the explanation utility for the user interface. He is now in the Advanced Development Group at Apple Computer.

DAVE PATTERSON is professor of computer science at the University of California at Berkeley, where he led the design of a VLSI microprocessor for Smalltalk, called SOAR (Smalltalk on a RISC). In 1982, he received a Distinguished Teaching Award from the university.

Cover design by Linda Petterson.



Norton

W • W • NORTON & COMPANY NEW YORK • LONDON
ISBN 0-393-95505-2