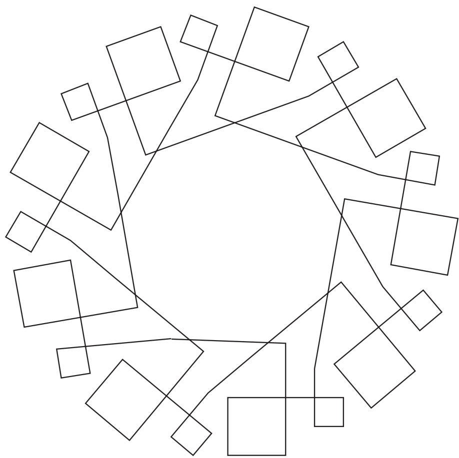
4916_Ch13_FINAL 5/18/05 3:59 PM Page 149

CHAPTER 13

Combining Methods



In Chapter 12, you learned how to define methods. I showed that defining methods is interesting and useful because (1) methods save you from having to rewrite scripts, which is time-consuming and subject to error, and (2) methods can be used and reused by different robots. The other main advantage of using methods is the possibility of using methods in other methods, that is, calling one or more existing methods as part of the definition of a new method. The reuse of methods is what we will explore in this chapter.

Being able to reuse methods is extremely important, because we can define a method in terms of another one without having to know all the details of how the second method is defined. We just call it and ask it to do what it is designed to do.

4916_Ch13_FINAL 5/18/05 3:59 PM Page 150

Nothing Really New: The Square Method Revisited

Having methods call other methods (which we call *composing methods*) is quite natural and is not really new. In fact, it is what you did in Chapter 12 when you defined a method! The method `square` includes in its definition calls to the methods `turnLeft:`, `go:`, and `timesRepeat:` (as shown in Method 13-1). Thus even the simple method `square` is defined in terms of other methods, and we did not have to know how `turnLeft:`, `go:`, and `timesRepeat:` are defined. We needed to know only what they do. So we are essentially done with this chapter, with nothing left to do but have some fun.

Method 13-1

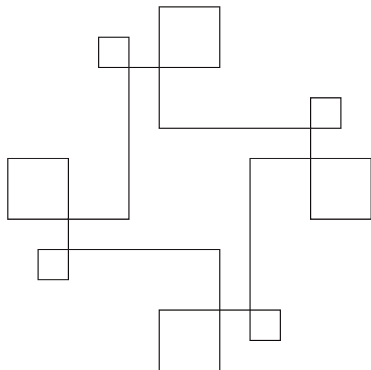
```
square
  "Draw a square of 100 pixels wide "

      4 timesRepeat:
      [ self go: 100;
             turnLeft: 90 ]
```

Other Graphical Patterns

In Chapter 12, I asked you to define the method `pattern`, which draws a simple abstract pattern (See script 12-5). Now I will ask you to perform some further experiments that will produce more drawings by defining more methods.

Experiment 13-1

Define a method `pattern4` that calls `pattern` four times to produce the figure below. You will use this method later, in another script. After you have created the method `pattern4`, use the following three-line script to make pica draw the figure:



```
| pica |
pica := Bot new.
pica pattern4
```

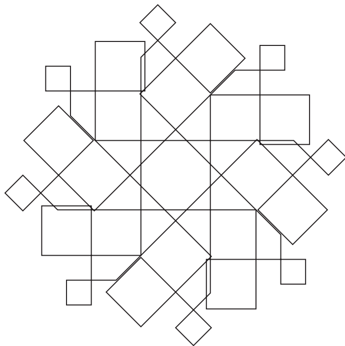
4916_Ch13_FINAL 5/18/05 3:59 PM Page 151

Experiment 13-2 (A Ferris Wheel)

Define a method called `tiltedPattern` that draws the picture at the beginning of this chapter, which looks somewhat like a Ferris wheel. Hint: you will have to call `pattern` nine times, and the angle through which to turn between calls is 10 degrees.

Experiment 13-3 (Doubling the Frame)

Define the method `doubleFrame`, presented below, that draws the picture shown after the method definition.



```
doubleFrame

   8 timesRepeat:
        [ self pattern.
        self turnLeft: 45.
        self go: 100 ]
```

What Do These Experiments Tell You?

Now let's see what you can learn from the experiments you did. As you can see from the methods `pattern4`, `tiltedPattern`, and `doubleFrame`, the method `pattern` was defined only once, and then *reused several times* in different methods. Defining `pattern` as a method allows you to (1) define it only once, (2) reuse it in various contexts, and (3) not introduce errors by copying this method over and over.

If you look at the definition of the method `doubleFrame`, you see that it is defined in terms of the `pattern` method, which is itself defined in terms of other methods, such as `go:` and `turnLeft:`. In fact, a complex method is often defined in terms of simpler methods, which themselves are defined in terms of even simpler methods, which themselves are defined in terms of even simpler methods, which themselves…. The advantage of this is that it is easier to understand and to define simple methods than complex methods, and the technique of defining methods in terms of simpler methods limits the degree of complexity in any one method. In Chapter 16, I will show you that to solve a problem, it is advantageous to decompose it into smaller subproblems, solve these subproblems, and then use the solutions to the smaller subproblems to solve the main problem.

4916_Ch13_FINAL 5/18/05 3:59 PM Page 152

It is essential to understand that in defining the method `doubleFrame`, you do not have to know how `pattern` is defined. You just need to know what it does and how to use it! When we define a method, we are giving a single name to a sequence of messages, which reduces the number of details that we have to keep track of. We just have to remember what the method does and its name, not how it does it. We say that we are building an *abstraction* over the definition details.

To make this point clear, I rewrote the method `doubleFrame` without calling the method `pattern` by directly copying the definition of `pattern` (shown in italics). Compare `doubleFrame➥WithoutCallingPattern` (Method 13-2) with the method `doubleFrame`. The new version without `pattern` is not only longer, but for most people it is also more confusing and harder to understand.

Now imagine what would happen if I did the same with the code of `turnRight:`, `turnLeft:`, and `go:`— because these are methods too. It would be a nightmare! There would be so many details that we would be lost all the time.

Method 13-2. *Creating the double frame without the abstraction of the pattern method*

```
doubleFrameWithoutCallingPattern

  8 timesRepeat:
      [ self go: 100.
      self turnRight: 90.
      self go: 100.
      self turnRight: 90.
      self go: 50.
      self turnRight: 90.
      self go: 50.
      self turnRight: 90.
      self go: 100.
      self turnRight: 90.
      self go: 25.
      self turnRight: 90.
      self go: 25.
      self turnRight: 90.
      self go: 50.
      self turnLeft: 45.
      self go: 100 ]
```

■ **Important!** When you write a new method, it can call other methods. You can use a method without knowing how it is written. After you finish writing a method, you can call it when you write another method.

4916_Ch13_FINAL 5/18/05 3:59 PM Page 153

Squares Everywhere

Now it is time to practice. Define the following methods using the method `square`.

Experiment 13-4 (Some Boxes)

Define methods `box` and `separatedBox` that produce the pictures shown in Figure 13-1.

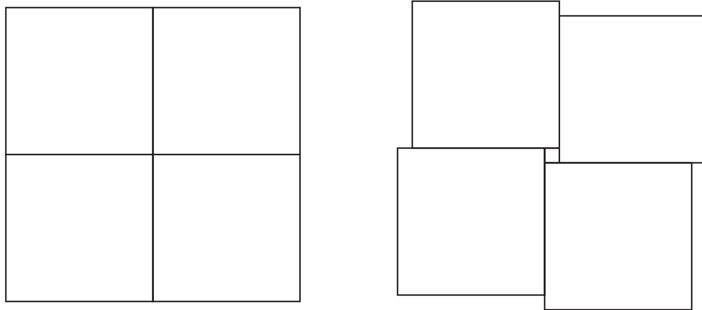


Figure 13-1. *Boxes*

Experiment 13-5 (Your Choice)

Use your previous methods to generate various figures of your choice. Have fun!

Experiment 13-6 (A Star)

Using the method `box`, experiment and define a method `star` that produces the right-hand picture in Figure 13-2.

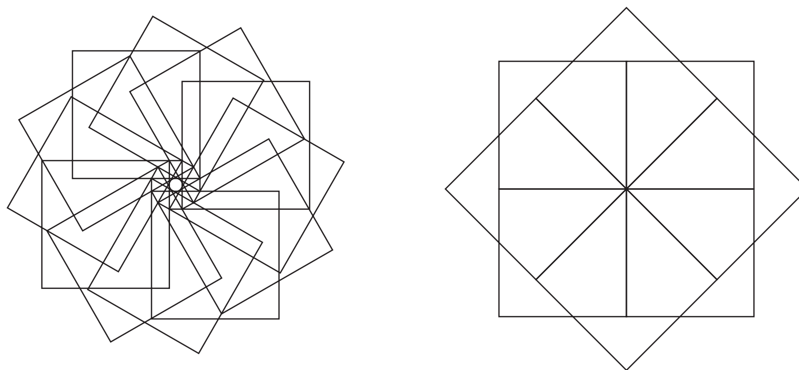
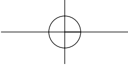


Figure 13-2. *Stars*

4916_Ch13_FINAL 5/18/05 3:59 PM Page 154



Summary

- When you write a new method, it can call other methods.
- You can use a method without knowing how it is written; you need to know only what it does.
- After you finish writing a method, you can call it when you write other methods.
- Hiding the details of a method by giving it a name is called *abstraction*.