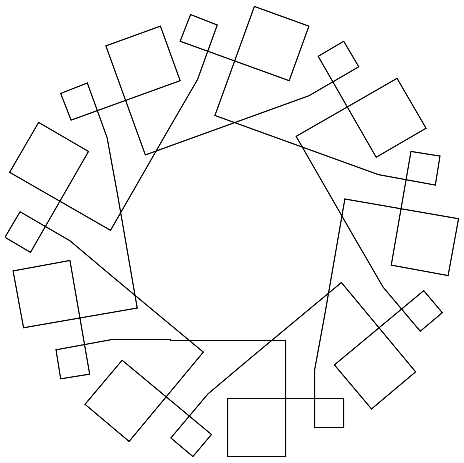
1
Combining Methods



In Chapter **??**, you learned how to define methods. I showed that defining methods is interesting because (1) methods avoid rewriting scripts and introducing errors and (2) methods can be used by different robots. The other main advantage of using methods is the possibility of reusing methods, that is defining a method by calling other already existing methods. Reuse 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.

1 Nothing Really New: the square Method

Composing methods is quite natural and is not really new. Actually it is what we did in Chapter **??** when you defined a method! The method square is defined by calling the methods turnLeft:, go: and timesRepeat: (as shown by method 1.1). Even square is defined in terms of other methods, and we don't have to know how turnLeft:, go: or timesRepeat: are defined. So we are already done with this chapter!

Method 1.1

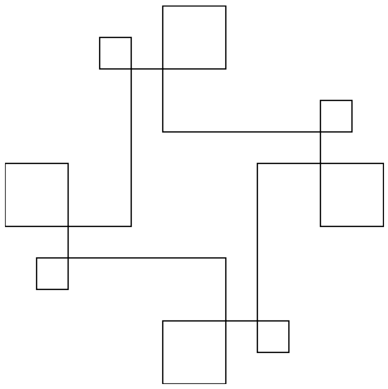
```
square
  "Draw a square of 100 pixels wide "

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

2 Other Graphical Patterns

In Chapter **??**, I asked you to define the method pattern that draws a nice pattern (See script **??**). Now I ask you to perform the experiment and produce the following drawings by defining more methods.

Experiment 1.1
Define the method pattern4 that calls pattern 4 times to produce the figure on the right. You will also use this method in another script later.

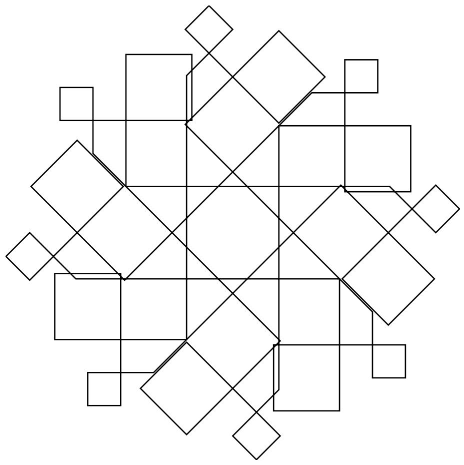
```
| pica |
pica := Bot new.
pica pattern4
```



Experiment 1.2
Define the method tiltedPattern draws the picture at the beginning of this Chapter. Hint: you have to repeat the pattern 9 times, and the angle to turn is 10 degrees.

Experiment 1.3
Define the method doubleFrame that draws the picture on the right.

```
doubleFrame

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



3 Stepping Back

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 is only defined 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 during the rewriting of this method.

If you look at the definition of the method doubleFrame, you see that it is defined in terms of the pattern method that is itself defined in terms of other methods such as go:, turnLeft:... and so on. In fact, a complex method is often defined in terms of simpler methods, which themselves are defined in terms of even simpler methods — and so on. This is because it is easier to understand and to define simpler methods. In Chapter **??** I will show you that to solve a problem it is simpler to decompose it into smaller subproblems, solve them and then use the solution to smaller subproblems to finally solve the first problem.

It is essential to understand that while defining the method doubleFrame we do not have to know how pattern is defined, we just need to know what it does and how to use it! When we define a method we are giving a name to a sequence of messages, which reduces the number of details that we have to remember. 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 italic) inside the other one. Compare doubleFrameWithoutCallingPattern (method 1.2) with the method doubleFrame. The new version without pattern is not only longer — 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 1.2

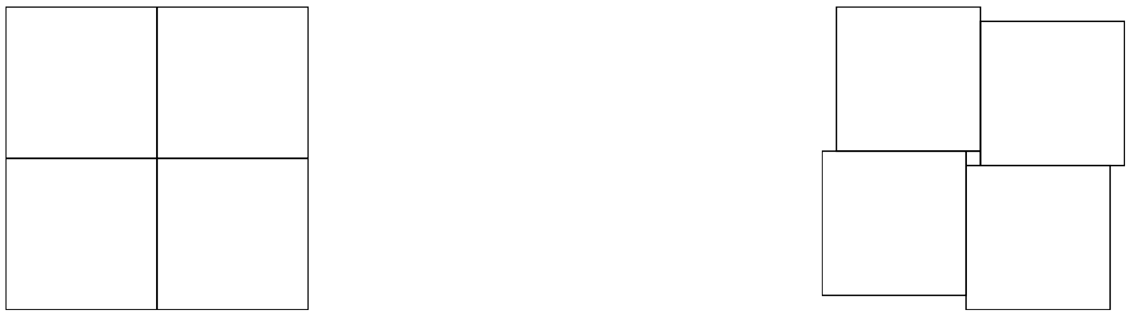
```
doubleFrameWithoutCallingPattern

  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 ]
```

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.

4 Squares Everywhere

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

Experiment 1.4
Some Boxes. Define the methods box and separatedBox that produce the pictures shown in Figure 4.



Experiment 1.5
Using your previous methods to generate various figures.

Experiment 1.6
Star. Using the method box, experiment and define a method star that produces the right-hand picture in Figure 4.

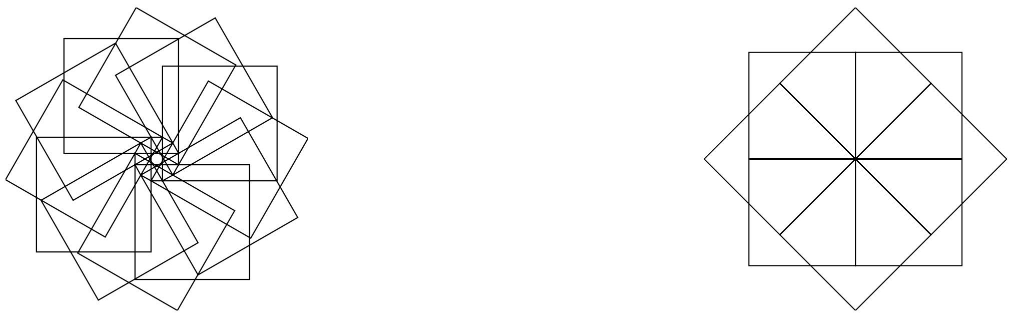


Figure 1.1: Stars

Summary

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.