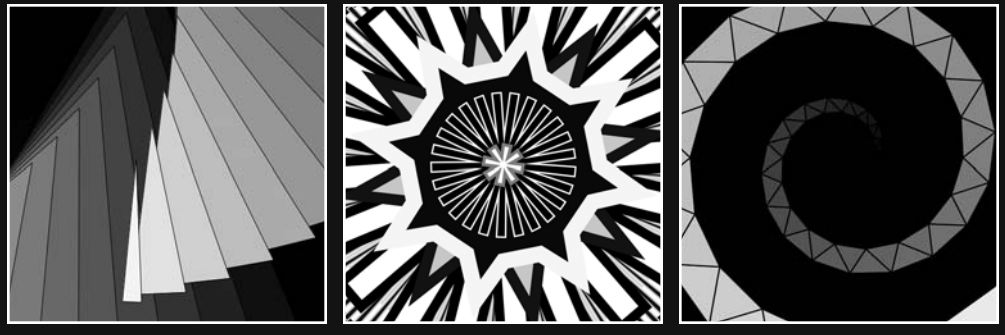
# 9 SHAPES



Shapes are a natural extension of lines and curves. In the simplest sense, a shape is just the enclosed space formed between three or more lines. Of course, a single continuous curve can enclose space as well (e.g., an ellipse). Shapes can be generally categorized. For example, an enclosed three-sided shape is referred to as a triangle, even if its size or configuration can't be determined. Fundamental 2D shapes (e.g., rectangles, circles, and triangles) are commonly referred to as **primitives**. Primitives are essential building-block shapes, which when combined allow you to (efficiently) construct most other shapes. Processing includes both 2D and 3D primitive functions. In this chapter, you'll just be dealing with 2D shape functions.

I'll begin by creating some simple sketches based on Processing's primitive 2D shape functions. These sketches will also provide a quick review of some of the core concepts covered in earlier chapters. Using Processing's drawing commands, I'll then show you how to create your own custom shapes, and eventually you'll work your way to implementing an OOP approach for shape creation—putting to good use some of the concepts you looked at in the last chapter. Before you dive in, first a word of encouragement.

## Patterns and principles (some encouragement)

Hopefully by this point in the book you're beginning to recognize some common coding and implementation patterns, duplicated in many of the examples. From my experiences in the classroom, I find most students are able to grasp these coding patterns and principles much more quickly than they can remember the specific language commands/syntax. You can always look up a command—that's what the language reference (API) is for. Conceptualization and design are far more important factors than language retention. Thus, if you're beginning to get the bigger picture, but are still struggling with implementation details, you're doing well. If on the other hand, fundamental coding concepts such as variables, loops, conditional statements, and arrays still seem unclear, you might want to review Chapter 3 again.

## Processing's shape functions

Processing comes with pre-made shape creation commands, some you've looked at already. However, I'll discuss each again with some simple examples. As you review these functions, I recommend trying to deduce the underlying algorithms operating within the functions, rather than simply trying to memorize how to apply them. For example, if you were to create your own rectangle-drawing function, how would you do it? This algorithm-centered approach will lead you to a deeper understanding of coding and Processing, and ultimately allow you to code any shape, rather than make you helplessly reliant on the few functions included in the API.

The first shape I'll cover is the rectangle. Using Processing, creating a rectangle couldn't be easier (and something you're probably very familiar with by now). The following code creates the rectangle shown in Figure 9-1: