Coding OOP

CHAPTER 13

Beauty is in the eye of the beholder.
Common paraphrasing of Plato

Coding is a personal endeavor. For many of us it's similar to creating art, and as with any art, there are many styles—some loved, others detested. Why is this relevant to coding? Well, because unlike the case with art, our code cannot stand alone. We are in the interesting position of creating art that, by definition, must work in a community.

No engineer intends to create complicated, stand-alone code. This chapter shows techniques, tricks, and idioms that you can use to communicate your intent. When your code is clear and transparent, other engineers can more easily understand, and appreciate, your intent. Code that is appreciated is more likely to be used appropriately, adapted, and, most important, integrated well with the rest of the system.

Overview

This chapter shows some of the coding techniques we can use to create our art. This, the last of the OOP chapters, talks about the coding going on inside a class. Of course, what's going on in a class is related to the class structures and interconnects around the code, so we will not limit our discussions to the lines of code in a method. Rather, we focus in this chapter on coding.

Our first focus is on "if" tests, with a discussion on why this necessary coding construct complicates the code. We'll show some ways to minimize these "if" tests.

We then discuss ways to get your point across, using coding tricks and idioms. We also look at the touchy subject of coding conventions, and try to point out where they help and where they hinder.

Finally, we look at templating and how it can be useful. The authors realize that we have covered templating in a few of the previous chapters, but here we take one last look at the C++ template library, as well as one more example of when to write your own template.

The previous part of this handbook used many different techniques, so we figured that one more look, concentrating on when to use individual techniques, might be useful.

"If" Tests—A Necessary Evil

The fewer "if" tests a segment of code has, the easier the code is to understand. Code that just does step one, followed by step two, and so on is inherently easier to reason about. An "if" test, by contrast, causes us to think some more. Is the condition true? Will the code set something up that I have to remember? Where was this condition set?

This section looks at ways to minimize "if" tests in your code. Of course, there will always be "if" tests in code; the goal is to find the place to put them so that their presence is reasonable and maybe even expected.