The Dependency Inversion Principle

It should be pretty clear that reducing dependencies to concrete classes in our code is a "good thing." In fact, we've got an OO design principle that formalizes this notion; it even has a big, formal name: *Dependency Inversion Principle*.

Yet another phrase you can use to impress the execs in the room! Your raise will more than offset the cost of this book, and you'll gain the admiration of your fellow developers.

Here's the general principle:

Design Principle

Depend upon abstractions. Do not depend upon concrete classes.

At first, this principle sounds a lot like "Program to an interface, not an implementation," right? It is similar; however, the Dependency Inversion Principle makes an even stronger statement about abstraction. It suggests that our high-level components should not depend on our low-level components; rather, they should *both* depend on abstractions.

A "high-level" component is a class with behavior defined in terms of other, "low-level" components.

For example, PizzaStore is a high-level component because its behavior is defined in terms of pizzas—it creates all the different pizza objects, and prepares, bakes, cuts, and boxes them, while the pizzas it uses are low-level components.

But what the heck does that mean?

Well, let's start by looking again at the pizza store diagram on the previous page. PizzaStore is our "high-level component" and the pizza implementations are our "low-level components," and clearly PizzaStore is dependent on the concrete pizza classes.

Now, this principle tells us we should instead write our code so that we are depending on abstractions, not concrete classes. That goes for both our high-level modules and our low-level modules.

But how do we do this? Let's think about how we'd apply this principle to our very dependent PizzaStore implementation...