**THE STRATEGY DESIGN PATTERN**

The GoF book says the Strategy design pattern should: “Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.”

The Strategy design pattern points out that, sometimes, it’s good to be taskoriented. That’s especially important if you want to maintain volatile code away from the main code for your app, or if you want to change the algorithm you use at runtime.

Consider using the Strategy design pattern if you have one of the following situations:

* You have volatile code that you can separate out of your application for easy maintenance.
* You want to avoid muddling how you handle a task by having to split implementation code over several inherited classes.
* You want to change the algorithm you use for a task at runtime.

So, there you have it — any time you start to get task-oriented and want to make those tasks one of the main design points of your code, the Strategy design pattern should spring to mind. That’s the way design patterns work.

They don’t provide you with specific code. Instead, you familiarize yourself with the idea, and when that idea could come in handy there’s an Aha! moment. This looks like a job for the Strategy pattern!

**THE DECORATOR DESIGN PATTERN**

In the Strategy Design Pattern, you encapsulate code in external algorithms for easy use rather than spreading it around inside your core code and modifying it throughout that code.The Decorator design pattern takes a different approach. Instead of using external algorithms, this design pattern is all about using *wrapper* code to

extend your core code.

The formal definition of the Decorator pattern from the GoF book (*Design Patterns: Elements of Reusable Object-Oriented Software*, 1995, Pearson Education, Inc. Publishing as Pearson Addison Wesley) says you can, “Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.”

This design pattern is called Decorator but that seems to imply optional frills. A better name for this pattern might be the “Augmentor” or “Extender” pattern because that’s what it allows you to do: augment or extend a class dynamically at runtime.