When is an accessor okay?

First, as I discussed earlier, it's okay for a method to return an object in terms of an interface that the object implements because that interface isolates you from changes to the implementing class. This sort of method (that returns an interface reference) is not really a "getter" in the sense of a method that just provides access to a field. If you change the provider's internal implementation, you just change the returned object's definition to accommodate the changes. You still protect the external code that uses the object through its interface.

Next, I think of all OO systems as having a procedural boundary layer. The vast majority of OO programs runs on procedural operating systems and talks to procedural databases. The interfaces to these external procedural subsystems are generic by nature. Java Database Connectivity (JDBC) designers don't have a clue about what you'll do with the database, so the class design must be unfocused and highly flexible. Normally, unnecessary flexibility is bad, but in these boundary APIs, the extra flexibility is unavoidable. These boundary-layer classes are loaded with accessor methods simply because the designers have no choice.

In fact, this not-knowing-how-it-will-be-used problem infuses all Java packages. It's difficult to eliminate all the accessors if you can't predict how you will use the class's objects. Given this constraint, Java's designers did a good job hiding as much implementation as they could. This is not to say that the design decisions that went into JDBC and its ilk apply to your code. They don't. We *do* know how we will use the classes, so you don't have to waste time building unnecessary flexibility.