languages in your head—Java plus the XML schema—instead of just Java? Besides, XML makes a clumsy programming language.

GUI builders had burned all of us before. Nobody wanted to end up with business logic woven into action listeners embedded in JPanels.

Reluctantly, we settled on a pure Swing GUI, but with some ground rules. Over a series of lunches at our local Applebee's, we hashed out a novel way of using Swing without getting mired in it.

UI and UI Model

The typical layered architecture goes "Presentation," "Domain," and "Persistence." In practice, the balance of code ends up in the presentation layer, the domain layer turns into anemic data containers, and the persistence layer devolves to calls into a framework.

At the same time, though, some important information gets duplicated up and down the layers. For instance, the maximum length of a last name will show up as a column width in the database, possibly a validation rule in the domain, and as a property setting on a `JTextField` in the UI.

At the same time, the presentation embeds logic such as "if *this* checkbox is selected, then enable *these* four other text fields." It sounds like a statement about the UI, but it really captures a bit of business logic: when the customer is a member of the Portrait Club, the application needs to capture their club number and expiration date.

So within the typical three-layer architecture, one type of information is spread out across layers, whereas another type of important information is stuck inside GUI control logic.

Ultimately, the answer is to invert the GUI's normal relationship to the domain layer. We put the domain in charge by separating the visual appearance of a screen from the logical manipulation of its values and properties.

Forms

In this model, a form object presents one or more domain objects' attributes as typed properties. The form manages the domain objects' lifecycles as well as calling down to the facades for transactions and persistence. Each form represents a complete screen full of interacting objects, though there are some limited cases where we use subforms.

The trick, though, is that a form is *completely* nonvisual. It doesn't deal with UI widgetry, only with objects, properties, and interactions among those properties. The UI can bind a Boolean property to any kind of UI representation and control gesture: checkbox, toggle button, text entry, or toggle switch. The form doesn't care. All it knows is that it has a property that can take a true/false value.