Forms never directly call screens. In fact, most of them don't even know the concrete class of their screens. All communication between forms and screens happens via properties and bindings.

### Properties

Unlike typical form-based applications, the properties that a `Form` exposes are not just Java primitives or basic types like `java.lang.Integer`. Instead, a `Property` contains a value together with metadata about the value. A `Property` can answer whether it is single-valued or multivalued, whether it allows null values, and whether it is enabled. It also allows listeners to register for changes.

The combination of `Forms` and their `Property` objects gave us a clean model of the user interface without yet dealing with the actual GUI widgetry. We called this layer the "UI Model" layer, as shown in Figure 4-4.

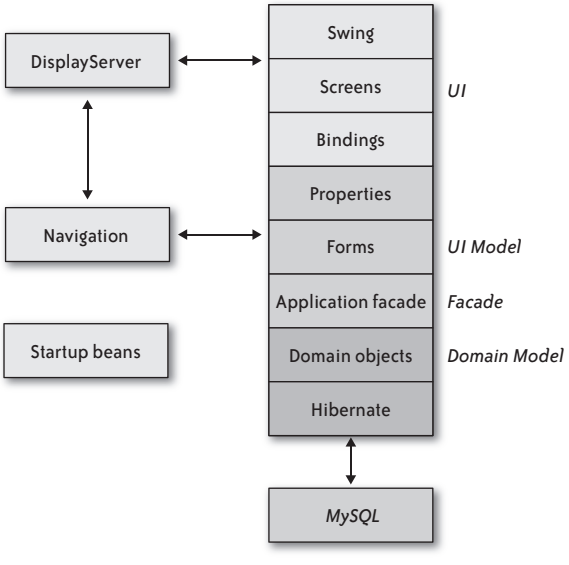


*FIGURE 4-4. Layered architecture*

Each subclass of `Property` works for a different type of value. Concrete subclasses have their own methods for accessing the value. For instance, `StringProperty` has `getStringValue()` and `setStringValue(String)`. Property values are always object types, not Java primitives, because primitives do not allow null values.

It might seem that property classes could proliferate endlessly. They certainly would if we created a property class for each domain object class. Most of the time, instead of exposing the domain object directly, the `Form` would expose multiple properties representing different