JAVABEANS

A Java Bean is a reusable software component that can be manipulated visually in a builder tool.

Beans will range greatly in their features and capabilities. Some will be very simple and others complex; some will have a visual aspect and others won't. Therefore, it isn't easy to put all Beans into a single category. Here are some of the most important features and issues surrounding Java Beans.

Properties, Methods, and Events

Properties are attributes of a Bean that are referenced by name. These properties are usually read and written by calling methods on the Bean specifically created for that purpose. A property of the thermostat component mentioned earlier in the chapter could be the comfort temperature. A programmer would set or get the value of this property through method calls, while an application developer using a visual development tool would manipulate the value of this property using a visual property editor.

The methods of a Bean are just the Java methods exposed by the class that implements the Bean. These methods represent the interface used to access and manipulate the component. Usually, the set of public methods defined by the class will map directly to the supported methods for the Bean, although the Bean developer can choose to expose only a subset of the public methods.

Events are the mechanism used by one component to send notifications to another. One component can register its interest in the events generated by another. Whenever the event occurs, the interested component will be notified by having one of its methods invoked. The process of registering interest in an event is carried out simply by calling the appropriate method on the component that is the source of the event. In turn, when an event occurs a method will be invoked on the component that registered its interest. In most cases, more than one component can register for event notifications from a single source. The component that is interested in event notifications is said to be *listening* for the event.

Introspection

Introspection is the process of exposing the properties, methods, and events that a JavaBean component supports. This process is used at run-time, as well as by a visual development tool at design-time. The default behavior of this process allows for the automatic introspection of any Bean. A low-level reflection mechanism is used to analyze the Bean’s class to determine its methods. Next it applies some simple design patterns to determine the properties and events that are supported. To take advantage of reflection, you only need to follow a coding style that matches the design pattern. This is an important feature of JavaBeans. It means that you don’t have to do anything more than code your methods using a simple convention. If you do, your Beans will automatically support introspection without you having to write any extra code. Design patterns are explained in more detail later in the chapter.

This technique may not be sufficient or suitable for every Bean. Instead, you can choose to implement a BeanInfo class which provides descriptive information about its associated Bean explicitly. This is obviously more work than using the default behavior, but it might be necessary to describe a complex Bean properly. It is important to note that the BeanInfo class is separate from the Bean that it is describing. This is done so that it is not necessary to carry the baggage of the BeanInfo within the Bean itself.

If you’re writing a development tool, an `Introspector` class is provided as part of the Beans class library. You don’t have to write the code to accomplish the analysis, and every tool vendor uses the same technique to analyze a Bean. This is important to us as programmers because we want to be able to choose our development tools and know that the properties, methods, and events that are exposed for a given component will always be the same.

Customization

When you are using a visual development tool to assemble components into applications, you will be presented with some sort of user interface for customizing Bean attributes. These attributes may affect the way the Bean operates or the way it looks on the screen. The application tool you use will be able to determine the properties that a Bean supports and build a property sheet dynamically. This property sheet will contain editors for each of the properties supported by the Bean, which you can use to customize the Bean to your liking. The Beans class library comes with a number of property editors for common types such as `float`, `boolean`, and `String`. If you are using custom classes for properties, you will have to create custom property editors to associate with them.

In some cases the default property sheet that is created by the development tool will not be good enough. You may be working with a Bean that is just too complex to customize easily using the default sheet. Beans developers have the option of creating a customizer that can help the user to customize an instance of their Bean. You can even create smart wizards that guide the user through the customization process.

Customizers are also kept separate from the Bean class so that it is not a burden to the Bean when it is not being customized. This idea of separation is a common theme in the JavaBeans architecture. A Bean class only has to implement the functionality it was designed for; all other supporting features are implemented separately.

Persistence

It is necessary that Beans support a large variety of storage mechanisms. This way, Beans can participate in the largest number of applications. The simplest way to support persistence is to take advantage of Java Object Serialization. This is an automatic mechanism for saving and restoring the state of an object. Java Object Serialization is the best way to make sure that your Beans are fully portable, because you take advantage of a standard feature supported by the core Java platform. This, however, is not always desirable. There may be cases where you want your Bean to use other file formats or mechanisms to save and restore state. In the future, JavaBeans will support an alternative externalization mechanism that will allow the Bean to have complete control of its persistence mechanism.

Design-Time vs. Run-Time

JavaBeans components must be able to operate properly in a running application as well as inside an application development environment. At design-time the component must provide the design information necessary to edit its properties and customize its behavior. It also has to expose its methods and events so that the design tool can write code that interacts with the Bean at run-time. And, of course, the Bean must support the run-time environment.

Visibility

There is no requirement that a Bean be visible at run-time. It is perfectly reasonable for a Bean to perform some function that does not require it to present an interface to the user; the Bean may be controlling access to a specific device or data feed. However, it is still necessary for this type of component to support the visual application builder. The component can have properties, methods, and events, have persistent state, and interact with other Beans in a larger application. An "invisible" run-time Bean may be shown visually in the application development tool, and may provide custom property editors and customizers.

Multithreading

The issue of multithreading is no different in JavaBeans than it is in conventional Java programming. The JavaBeans architecture doesn't introduce any new language constructs or classes to deal with threading. You have to assume that your code will be used in a multithreaded application. It is your responsibility to make sure your Beans are thread-safe. Java makes this easier than in most languages, but it still requires some careful planning to get it right. Remember, thread-safe means that your Bean has anticipated its use by more than one thread at a time and has handled the situation properly.

Security

Beans are subjected to the same security model as standard Java programs. You should assume that your Bean is running in an untrusted applet. You shouldn't make any design decisions that require your Bean to be run in a trusted environment. Your Bean may be downloaded from the World Wide Web into your browser as part of someone else's applet. All of the security restrictions apply to Beans, such as denying access to the local file system, and limiting socket connections to the host system from which the applet was downloaded.

If your Bean is intended to run only in a Java application on a single computer, the Java security constraints do not apply. In this case you might allow your Bean to behave differently. Be careful, because the assumptions you make about security could render your Bean useless in a networked environment.