The Java Beans Design Pattern

Jeff Offutt

http://www.cs.gmu.edu/~offutt/

SWE 642
Software Engineering for the World Wide Web

Java Server Pages and Beans

• Use JSPs to define look of web pages, and fill them with information from Java Beans
• This helps separate presentation (HTML) from content (variables)
• Beans are based on
  – Events
  – Properties
  – Methods
• The JavaBeans API specification:
Java Beans Purpose

- Java Beans provide design patterns to:
  - Set and retrieve properties
  - Pass events between objects
  - Create instances of objects
  - Store objects using serialization (lists)
- Also possible to describe information about the bean (meta-info)
- The concepts have developed from graphical programming
  - Specifically from the event-notify pattern

Java Beans and Properties

Web programming often uses properties
- From the outside: properties store and retrieve values
- From the inside: values can be stored in variables, files, databases, computed, or retrieved from another object
- Properties have two common accessor methods:
  - public void setAge(int age)
  - public int getAge()
- Java introspection: given a name (age) and a type (int), accessor methods are assumed to exist
- JSP syntax—two equivalent types of statements
  1. `<jsp:getProperty id="user" property="age" />`
  2. `<%= user.getAge() %>`
Java Bean Overview

- A component is a reusable software building block:
  - A pre-built piece of encapsulated application code that can be combined with other components and with handwritten code to rapidly produce a custom application
  - There are many definitions … mostly fairly similar

- A component architecture defines
  - How a collection of components should divide up functions
  - How the components should communicate

- The J2EE platform defines JavaBeans for component architectures
  - Conventions for how software components interact
  - Some built-in Java classes

Example Component Architecture

Photo Personal Web Application

client -> servlet

get form

submit

client -> JSP

get data

Model I Architecture

I/O

file store
What are JavaBeans?

- Definition: “A JavaBean component is an object that conforms to a communication and configuration protocol, as prescribed by the JavaBeans specification”
- Standard Java classes versus beans:
  - A Bean is a class or a group of classes that follows the naming convention of the JavaBeans API
  - This allows introspection
- Predefined design patterns to relieve developers from the burden of low and mid-level design
- Ultimately an outgrowth of data abstraction and ADTs
Java Bean Technologies

Five key technologies support the design pattern:
1. Bean events and event handling
2. Bean properties
3. Introspection
4. Customization
5. Persistence

1: Events

• Beans are interconnected via events
• The event model is adopted from the JDK 1.1 AWT event model
• Goals :
  – Make it easier to connect methods to events
  – Use strong typing
  – Use standard “design patterns”
• Event firing
  – Beans must allow “listeners” to register
  – An event “fires” by invoking a named, typed method on a listener (another Java class)
1: Event Handling

- Two types of event sources:
  - Unicast: Allows only one listener to register for events
  - Multicast: Allows multiple listeners to register for events

Beans are interconnected by using JDK event handling
2: Bean Properties

- **Bean property**:  
  - A named attribute of a Bean that can affect its behavior or its appearance  
  - To speak plainly: A variable
- **Accessed via getter / setter methods**
- **Beans can be connected using properties and methods**

2: Bean Properties

- **Types of Bean Properties**:
  1. **Simple**: A single value whose changes are independent of any other property
  2. **Bound**: A change requires another Bean to be notified  
     - Example: Change to *address* requires *phone number* to be changed
  3. **Constrained ("vetoable")**: A change must be *validated* by another Bean  
     - The change may be *rejected*
     - Example: A *password* can be changed only if *current user* is authorized
  4. **Indexed**: A range of values instead of a single value is supported
2: Bean Properties

• Properties can be read-only, write-only, or read-write, according to the accessor methods

• Accessor Methods:
  The name of the Bean property and the names of the property accessor methods depend on each other

• For a simple read-write property of type Clock:
  public Clock getHour () ;
  public void setHour (Clock p) ;

• Boolean properties:
  isAlarm ()
  setAlarm ()

• Indexed properties:
  public alarmTime getAlarm (int index) ;
  public void setAlarm (int index, alarmTime p) ;
  public alarmTime [ ] getAlarm () ;
  public void setTime (alarmAlarm [ ] p) ;
2: Implementing Bean Properties

Defining simple properties

Bean ColoredBean with one simple property color :

```java
public class ColoredBean {
    private Color myColor = Color.black;
    public Color getColor () {
        return myColor;
    }
    public void setColor (Color newColor) {
        myColor = newColor;
    }
} // ColoredBean
```

Defining bound properties

• Accessor methods are defined just as for simple properties

• Additionally the Bean has to have:
  `PropertyChangeEvents`
  – This will notify `PropertyChangeListener` by calling their `propertyChange` methods
2: Implementing Bean Properties

Defining constrained properties

• The set method is written as follows:
  – Save the old value
  – Notify registered *VetoableChangeListener*s
  – If no listeners veto, set the property to the new value, else
    throw a *PropertyVetoException*

• When changing the value of a constrained property,
  the source Bean must:
  – catch exceptions
  – eventually revert the old value, and
  – notify all listeners of the reversion

3: Introspection

• For an application to use a Bean, the application must
  have certain information about the Bean
• Information must be platform independent and software-readable
• Introspection allows the Beans *properties*, *events* and
  *methods* to be discovered
• Assumes the standard design conventions are followed
• Explicit information:
  – developers can give specific information about a Bean by
    supplying an object whose class implements
    *java.beans.BeanInfo*
3: Introspection
Uses for BeanInfo

Reasons for explicitly adding BeanInfo

- Limit a long list of properties or events to a few important ones
- Provide a GIF image as an icon for the builder’s component palette
- Add descriptive, human readable, and possibly localized names for properties
- Make properties “hidden” or “expert” to accommodate different models of development

4: Bean Customization

- Beans expose properties so they can be customized at design time
- Customization allows designers to modify Bean properties within an application builder
- Properties are edited through property sheets
- Customization is supported in two ways:
  - using property editors
  - using more sophisticated Bean customizers
5: Bean Persistence

- Bean properties can be stored to disk
- How to store them?
  - Make the Bean object persistent
  - Java supports persistence through serialization
- Serialization:
  - Classes must implement the `Serializable` interface
  - Serializable objects can be written to and read from a stream
  - Example: Store to and read from a file
- By default, serialization includes all fields of an object
  - Fields that must be excluded have to be marked `transient` or `static`

Developing with Beans

- Standard Components
  - Use “off the shelf”
- Builder Tools
  - Customize, connect
  - Create new components
- Developer
  - Ship as
- Application
JavaBeans Summary

- JavaBeans provide the component architecture of Java
- The technologies JavaBeans builds upon are existing Java features:
  - Object orientation
  - Reflection
  - Serialization
- Beans can be created easily
  - Existing programs are often already programmed in a “Bean-like” way