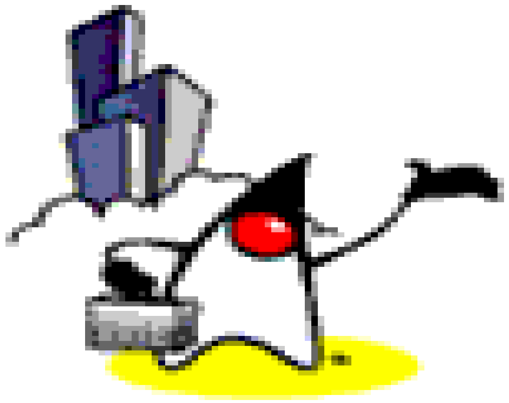
JavaBeans
(Importance scale: ****, Very important)

Sang Shin
Michèle Garoche
www.javapassion.com
“Learn with Passion!”

Topics

- JavaBean as a software component model
- Core concepts of JavaBeans
- Properties
- Event model
- Introspection
- Bean persistence
- Bean persistence in XML



JavaBean as a Software Component Model

What is a Software “Component”?

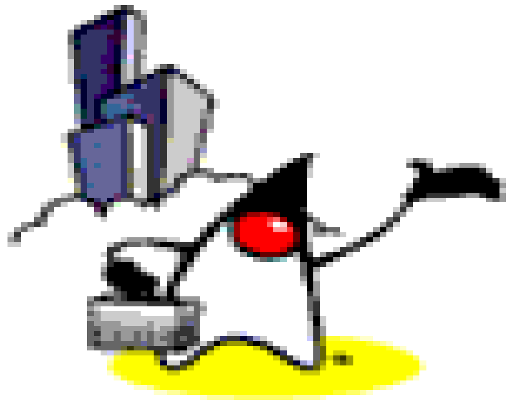
- Software components are self-contained, reusable software units
- Software components can be categorized into two
  - Visual or Non-Visual components
- Visual software components
  - Readily available to UI builder tools - Button, TextView
  - Can be dragged and dropped to build UI of an application
  - You can immediately see the results of your work - for example changing a color property of a component
- Non-visual software components
  - Capture business logic or state
  - Product, Customer

What is a JavaBean?

- JavaBeans™ is a portable, platform-independent component model written in the Java programming language
- With the JavaBeans API you can create reusable, platform-independent components
- Can be visual or non-visual
- JavaBeans (non-Visual ones) are the basis of POJO (Plain Old Java Object) based programming

What is a JavaBean?

- JavaBean components are sometimes called just beans.
- Through the design mode of a UI builder tool, you use the property sheet or bean customizer to customize the bean and then save (persist) your customized beans.
- Beans are also dynamic in that the properties can be changed during runtime



Core Concepts of JavaBeans

UI Builder Tools & Introspection

- UI Builder tools discover a bean's features (that is, its properties, methods, and events) by a process known as introspection.
- Beans support introspection in two ways:
  - By adhering to specific rules, known as design patterns, when naming bean features, for example, a property named *color* can be exposed through getColor() and *setColor()* methods
  - By explicitly providing property, method, and event information with a related bean information class (This is rarely used, thus is less important)

Properties of a Bean

- Properties represent characteristics of a bean
  - Could be visual such as *color* of a *Button* bean or non-visual such as *balance* of *SavingsAccount* bean
- Beans expose properties through accessor methods so they can be set/get
  - Accessor methods are getXXX() and setXXX() methods
- UI Builder tools introspect on a bean to discover its properties and expose those properties for manipulation
  - It checks if the bean has getXXX() and setXXX() methods

Events

- Beans use events to communicate with other beans
  - Button bean reacts to “Button clicked” event
- A bean that is to receive events (a event listener bean) registers with the bean that fires the event (a event source bean)
- UI Builder tools can examine a bean and determine which events that bean can fire (send) and which it can handle (receive)
  - It checks if a Button bean has event handler method associated with it

Persistence

- Persistence enables beans to save and restore their state (values of its properties)
- After changing a bean's properties, you can save the state of the bean and restore that bean at a later time
- The JavaBeans architecture uses Java Object Serialization to support persistence.

JavaBean Method

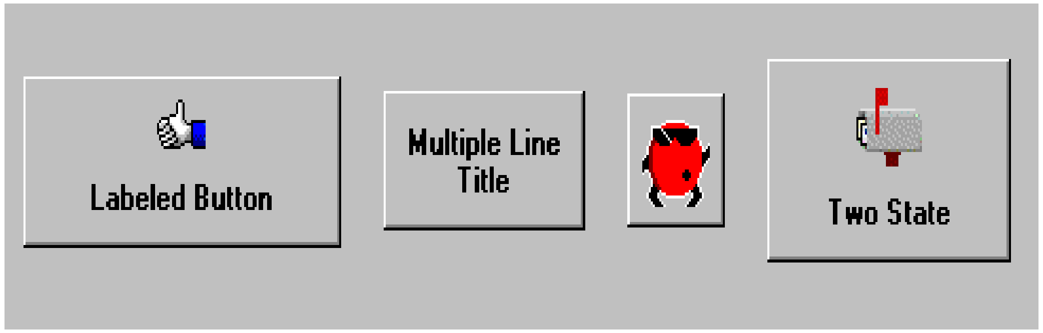
- A bean's methods are no different from Java methods, and can be called from other beans or a scripting environment
- By default all public methods are exported

Examples of Beans

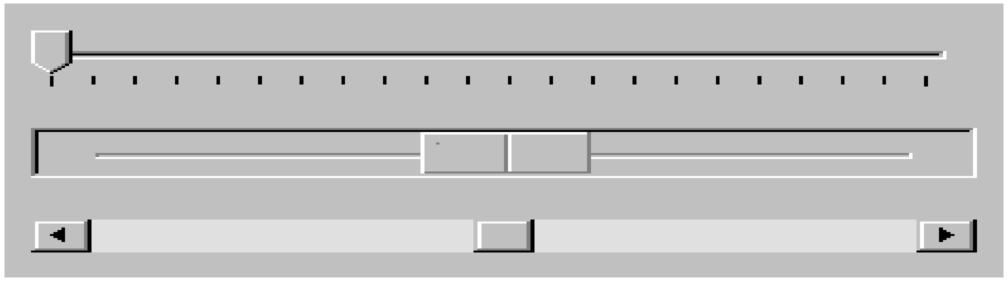
- Visual GUI (graphical user interface) component
  - Button, Slider, TextEdit, DropDown
- Non-visual beans
  - SavingsAccount
  - Spelling checker
  - ...
  - Pretty much all Java classes with getXXX()/setXXX() methods are considered as Beans

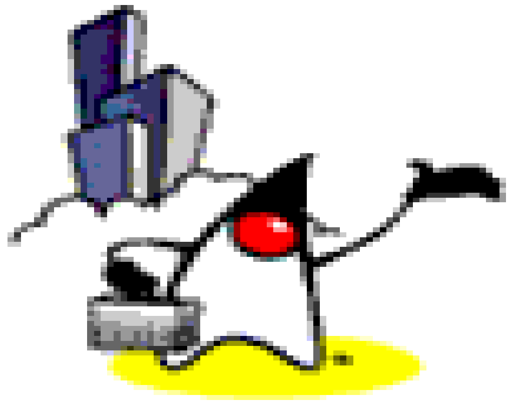
Examples of GUI Beans

- Button Beans



- Slider Bean





Event Model (Important)

Players in the JavaBeans Event Model

- Event source
  - An object that generates (fire) events
  - Sometimes called Event sender or Event generator
  - Button bean is example of an Event source
- Event listener
  - An object interested in receiving events
  - Sometimes called Event handler or Event receiver
  - Business logic bean - for example, when “Delete file” button is clicked, a business logic that actually performs file-deleting is an event listener

How Does Event Source Know the Event Listeners?

- Event listeners register their interest of receiving events to the event source
  - Event source class provides the methods for event listeners to register and unregister
- The event source maintains a list of event listeners who registered and invoke them when an event occurs

Registration of Event Listeners

- Event source provides methods that are used by the event listeners to register
- The methods are typically as following - XXX is the name of the Event
  - addXXXListener(..)
  - removeXXXListener(..)
- Example: Suppose you are writing a GUI application
  - You might have Button object, which functions as an event source
  - Button class might provide the following methods
    - addOnClickListener(..)
    - removeOnClickListener(..)

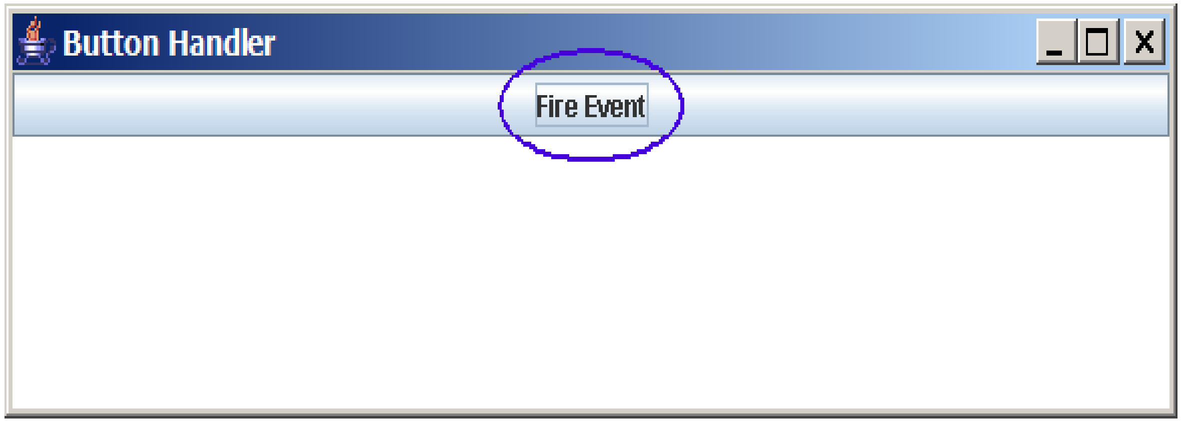
Steps of Writing Custom Event Handling

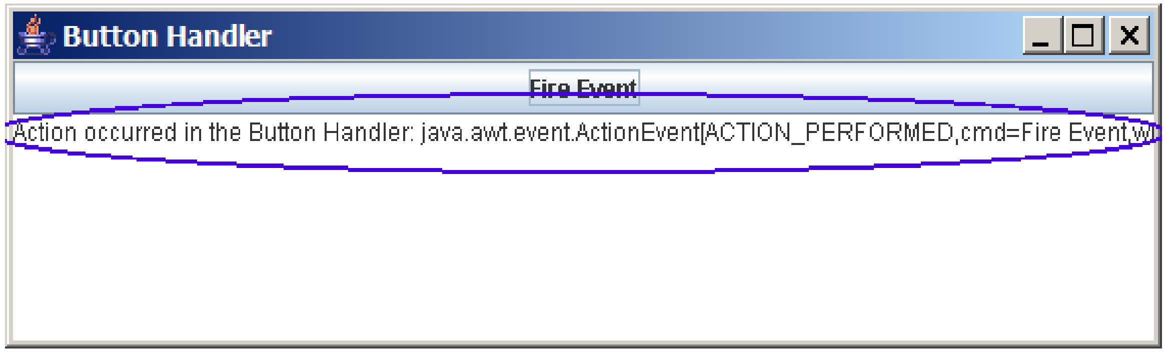
1. Write Event class
   - Create your own custom event class, named XXXEvent or use an existing event class either from JDK (i.e. ActionEvent) or from someone else
2. Write Event listener interface and implementation
   - Write XXXListener interface and provide implementation class of it
   - There are built-in listener interfaces (i.e. ActionListener)
3. Write Event source
   - Add an addXXXListener(..) and removeXXXListener(..) methods, where XXX stands for the name of the event
   - These methods are used by event listeners for registration

Steps of Adding Event Handling

4. Write a glue class
   - Register event listener to the event source through addXXXListener() method of the event source

Example 1: Button Handler





```
Action occurred in the Button Handler: java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=Fire Event,w
```

1. Write Event Class

- We are going to use *ActionEvent* class which is already provided in JDK

2. Write Event Listener Interface and its Implementation Class

- We are going to use *ActionListener* interface which is already provided in JDK
  - The interface already has *actionPerformed(ActionEvent event)* method, which needs to be implemented
- We are going to write our own *ButtonHandler* class which implements *ActionListener* interface
  - We provide the some logic in the implementation of the *actionPerformed(ActionEvent event)* abstract method

2. Write Event Listener Class

```
public class ButtonEventHandler implements ActionListener {
    /**
     * Component that will contain messages about
     * events generated.
     */
    private JTextArea jTextArea;
    /**
     * Creates an ActionListener that will put messages in
     * JTextArea everytime event received.
     */
    public ButtonHandler( JTextArea jTextArea ) {
        this.jTextArea = jTextArea;
    }

    /**
     * When receives action event notification, appends
     * message to the JTextArea passed into the constructor.
     */
    public void actionPerformed( ActionEvent event ) {
        this.jTextArea.append( "Action occurred in the Button Handler: " + event + '\n' );
    }
```

3. Write Event Source Class

- We are going to use *Button* class which is event source class and is already provided in JDK
- *The* JDK-provided *Button* class already has the following methods (that can be used by event listeners for registration/unregistration)
  - addActionListener
  - removeActionListener

4. Write Glue Code

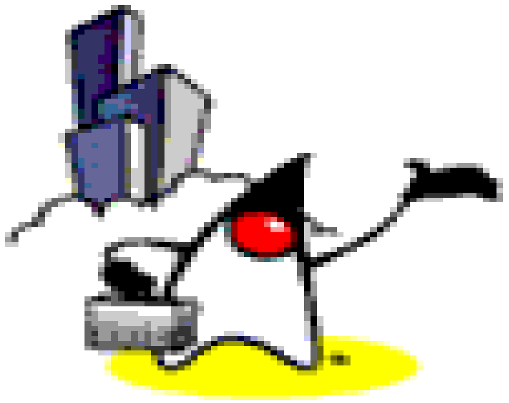
- Create object instances
- Register event handler to the event source

4. Write Glue Code

```
public class ActionEventExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame( "Button Handler" );
        JTextArea area = new JTextArea( 6, 80 );

        // Create event source object
        JButton button = new JButton( "Fire Event" );

        // Register an ActionListener object to the event source
        button.addActionListener( new ButtonEventHandler( area ) );

        frame.add( button, BorderLayout.NORTH );
        frame.add( area, BorderLayout.CENTER );
        frame.pack();
        frame.setDefaultCloseOperation( WindowConstants.DISPOSE_ON_CLOSE );
        frame.setLocationRelativeTo( null );
        frame.setVisible( true );
    }
```

What Happens When an Event Occurs?

- Event source invokes event handling method of all Event handlers (event listener) registered to it
  - *actionPerformed()* method *ButtonEventHandler* will be invoked



Introspection

What is Introspection?

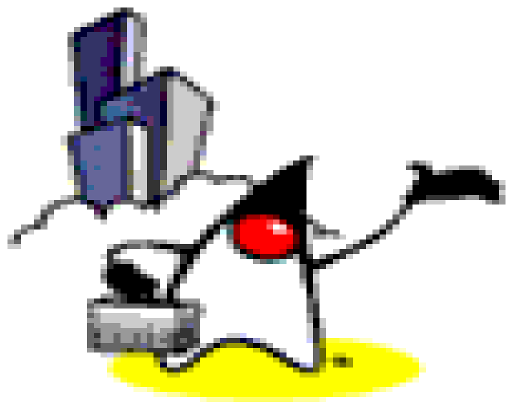
- Introspection is the process of analyzing a bean's design patterns to reveal the bean's properties, events, and methods
  - This process controls the publishing and discovery of bean operations and properties
- By default, introspection is supported by reflection, where you name methods with certain naming patterns, like setXxx/getXxx() where Xxx is the name of the property and addYyyListener()/removeYyyListener() where Yyy is the name of the event

Things That Can be Found through Introspection

- Simple property
  - public void setPropertyName(PropertyType value);
  - public PropertyType getPropertyName();
- Boolean property
  - public void setPropertyName(boolean value);
  - public boolean isPropertyName();
- Indexed property
  - public void setPropertyName(int index, PropertyType value);
  - public PropertyType getPropertyName(int index);
  - public void setPropertyName(PropertyType[] value);
  - public PropertyType[] getPropertyName();

Things That can be found through Introspection

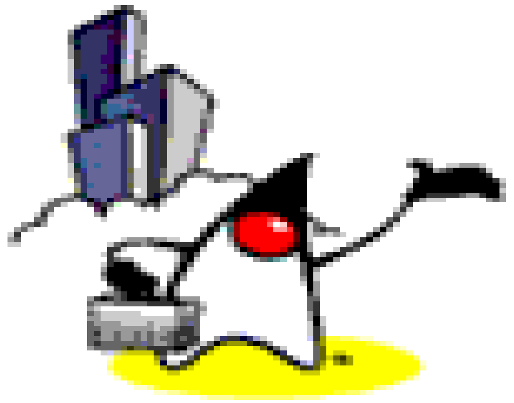
- Multicast events
  - public void addEventListenerType(EventListenerType l);
  - public void removeEventListenerType(EventListenerType l);
- Unicast events
  - public void addEventListenerType(EventListenerType l) throws TooManyListenersException;
  - public void removeEventListenerType(EventListenerType l);
- Methods
  - public methods



BeanInfo Interface

What is BeanInfo Interface?

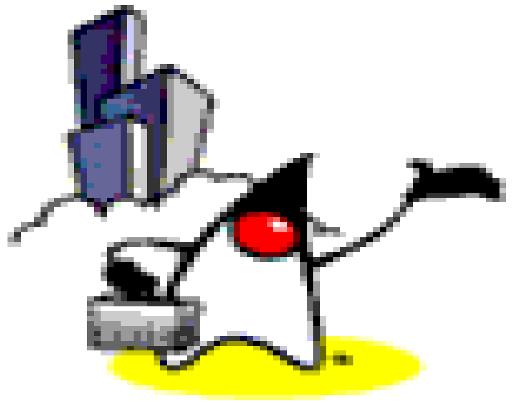
- It is an interface of the java.beans package that defines a set of methods that allow bean implementors to provide explicit information about their beans.
- By providing implementation class of the BeanInfo interface for a bean component, a developer can hide methods, specify an icon for the toolbox, provide descriptive names for properties, and much more



Bean Persistence

Bean Persistence

- Through object serialization
  - Object serialization means converting an object into a data stream and writing it to storage.
- Any applet, application, or tool that uses that bean can then "reconstitute" it by deserialization. The object is then restored to its original state
- For example, a Java application can serialize a Frame window object on a Windows platform, the serialized file can be sent with e-mail to a Linux platform, and then a Java application can restore the Frame window object to the exact state which existed on the Windows platform.



Bean Persistence in XML

XMLEncoder Class

- Enable beans to be saved in XML format
- The *XMLEncoder* class is assigned to write output files for textual representation of Serializable objects

```
XMLEncoder encoder = new XMLEncoder(
              new BufferedOutputStream(
                     new FileOutputStream( "Beanarchive.xml" ) ) );

encoder.writeObject( object );
encoder.close();
```

XMLDecoder Class

- XMLDecoder class reads an XML document that was created with XMLEncoder:

```
XMLDecoder decoder = new XMLDecoder(
                new BufferedInputStream(
                        new FileInputStream( "Beanarchive.xml" ) ) );

Object object = decoder.readObject();
decoder.close();
```

Example: SimpleBean

```
import java.awt.Color;
import java.beans.XMLDecoder;
import javax.swing.JLabel;
import java.io.Serializable;

public class SimpleBean extends JLabel
                        implements Serializable {
    public SimpleBean() {
        setText( "Hello world!" );
        setOpaque( true );
        setBackground( Color.RED );
        setForeground( Color.YELLOW );
        setVerticalAlignment( CENTER );
        setHorizontalAlignment( CENTER );
    }
}
```

Example: XML Representation

```
<?xml version="1.0" encoding="UTF-8" ?>
<java>
  <object class="javax.swing.JFrame">
    <void method="add">
      <object class="java.awt.BorderLayout" field="CENTER"/>
      <object class="SimpleBean"/>
    </void>
    <void property="defaultCloseOperation">
      <object class="javax.swing.WindowConstants"
    field="DISPOSE_ON_CLOSE"/>
    </void>
    <void method="pack"/>
    <void property="visible">
      <boolean>true</boolean>
    </void>
  </object>
</java>
```

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"