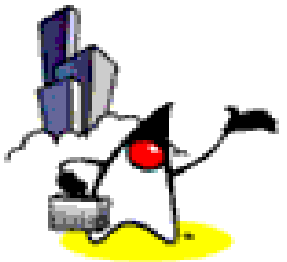
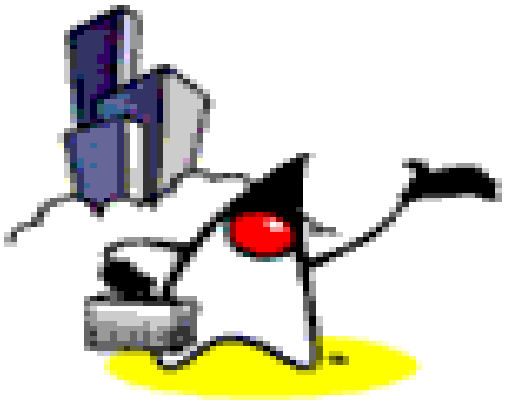
JavaBeans



Topics

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



JavaBean as a Software Component Model

Software Component

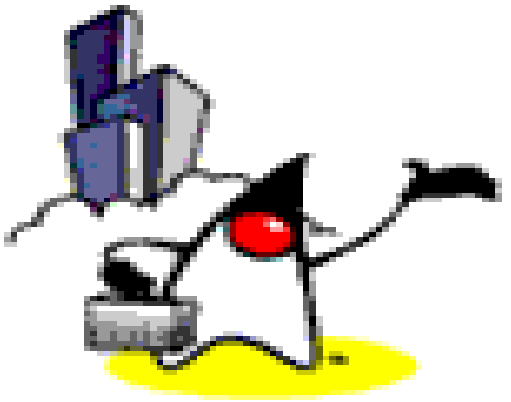
- Software components are self-contained, reusable software units
- Visual software components
  - Using visual application builder tools, visual software components can be composed into applets, applications, servlets, and composite components
  - You perform this composition within a graphical user interface, and you can immediately see the results of your work.
- Non-visual software components
  - Capture business logic or state

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
- Using JavaBeans-compliant application builder tools such as NetBeans or Eclipse, you can combine these components into applets, applications, or composite components.

What is a JavaBean?

- JavaBean components are known as beans.
- Beans are dynamic in that they can be changed or customized
- Through the design mode of a builder tool, you use the property sheet or bean customizer to customize the bean and then save (persist) your customized beans.



Core Concepts of JavaBeans

Builder Tools & Introspection

- 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
  - By explicitly providing property, method, and event information with a related bean information class.

Properties

- Properties are the appearance and behavior characteristics of a bean that can be changed at design time
- Beans expose properties so they can be customized at design time
- Builder tools introspect on a bean to discover its properties and expose those properties for manipulation
- Customization is supported in two ways:
  - by using property editors
  - by using more sophisticated bean customizers

Events

- Beans use events to communicate with other beans
- A bean that is to receive events (a listener bean) registers with the bean that fires the event (a source bean)
- Builder tools can examine a bean and determine which events that bean can fire (send) and which it can handle (receive)

Persistence

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

JavaBean Method

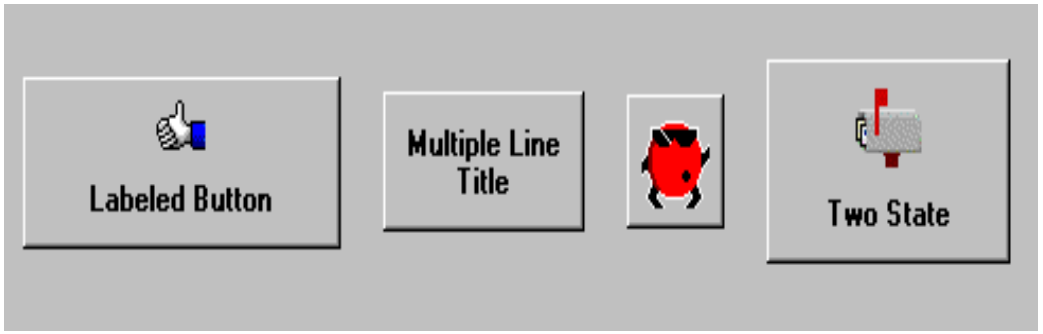
- A bean's methods are no different from Java methods, and can be called from other beans or a scripting environment
- By default all public methods are exported

Examples of Beans

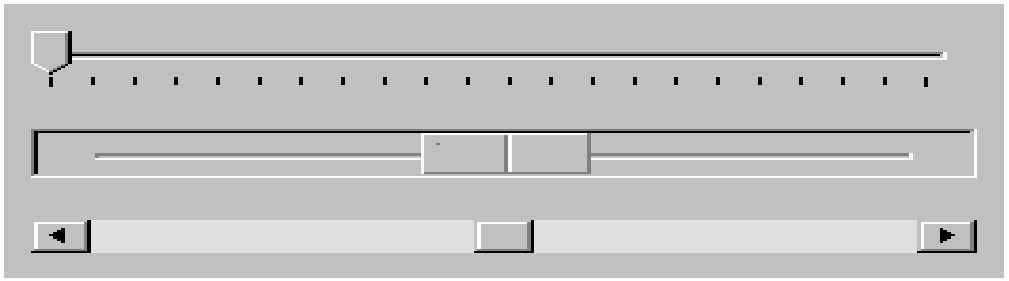
- GUI (graphical user interface) component
- Non-visual beans, such as a spelling checker
- Animation applet
- Spreadsheet application

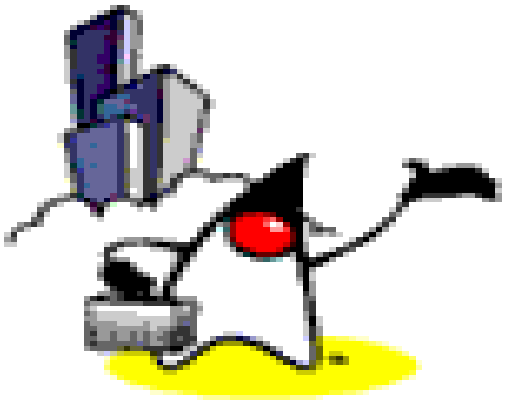
Examples of GUI Beans

- Button Beans



- Slider Bean





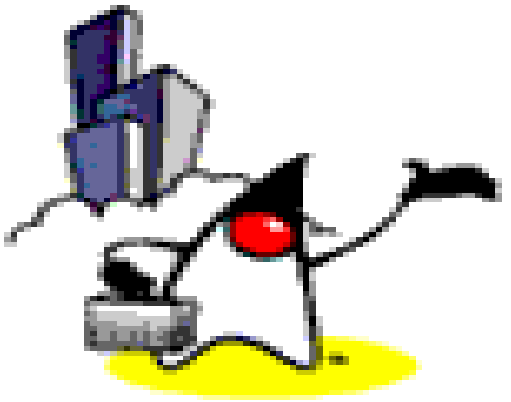
Properties

Properties

- A bean property is a named attribute of a bean that can affect its behavior or appearance
- Examples of bean properties include color, label, font, font size, and display size.

Types of Properties

- Simple – A bean property with a single value whose changes are independent of changes in any other property.
- Indexed – A bean property that supports a range of values instead of a single value.
- Bound – A bean property for which a change to the property results in a notification being sent to some other bean.
- Constrained – A bean property for which a change to the property results in validation by another bean. The other bean may reject the change if it is not appropriate.



Event Model

JavaBeans Event Model

- Based the Java 1.1 event model
- An object interested in receiving events is an event listener – sometimes called event receiver
- An object that generates (fire) events is called an event source – sometimes called event sender
- Event listeners register their interest of receiving events to the event source
  - Event source provides the methods for event listeners to call for registration
- The event source maintains a list of listeners and invoke them when an event occurs

Registration of Event Listeners

- Event listeners are registered to the event source through the methods provided by the event source
  - addXXXListener
  - removeXXXListener

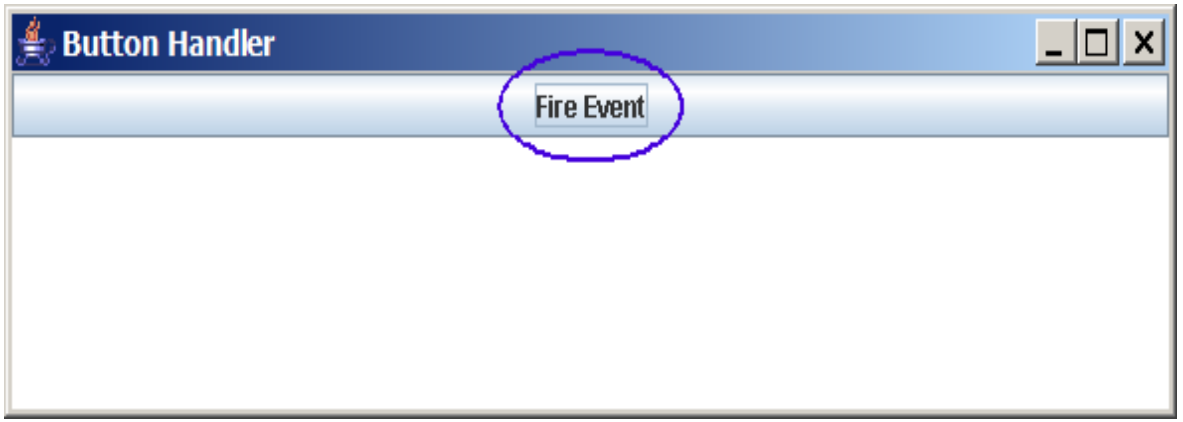
Steps of Writing Event Handling

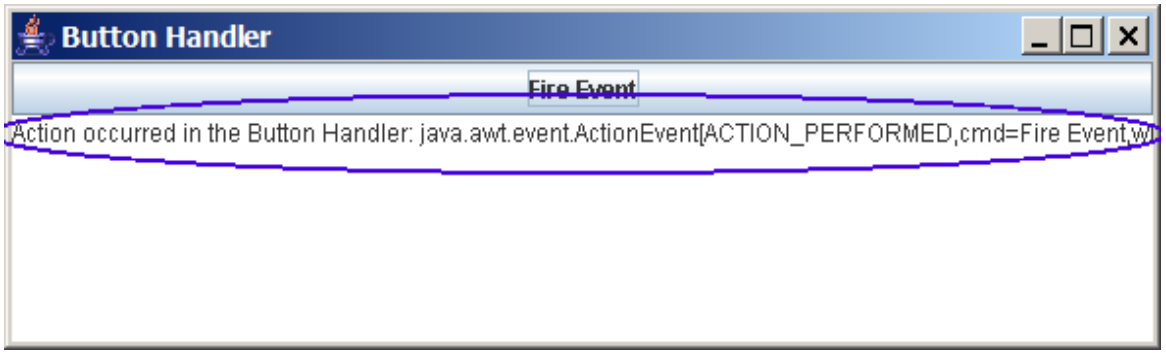
1. Write Event class
   - Create your own custom event class, named XXXEvent or use an existing event class
   - There are existing event class (i.e. ActionEvent)
2. Write Event listener (Event handler or Event receiver)
   - Write XXXListener interface and provide implementation class of it
   - There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

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





1. Write Event Class

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

2. Write Event Listener Class

- We are going to use *ActionListener* interface which is already provided in JDK
- We are going to write *ButtonHandler* class which implements *ActionListener* interface

2. Write Event Listener Class

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

    /**
     * When receives action event notification, appends
     * message to the JTextArea passed into the constructor.
     */
    public void actionPerformed( ActionEvent event ) {
        this.output.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
- Button class already has the following methods
  - addActionListener
  - removeActionListener

4. Write Glue Code

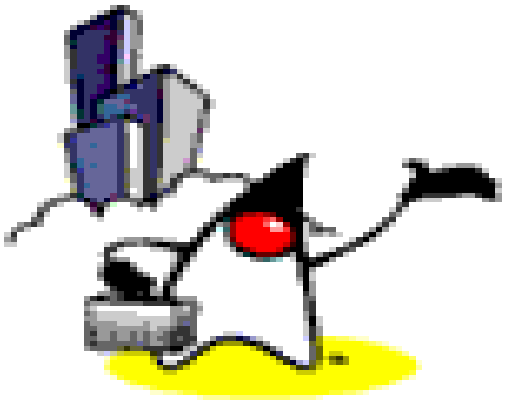
- Create object instances
- Register event handler to the event source

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 ButtonHandler( area ) );

        frame.add( button, BorderLayout.NORTH );
        frame.add( area, BorderLayout.CENTER );
        frame.pack();
        frame.setDefaultCloseOperation( WindowConstants.DISPOSE_ON_CLOS
        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 ButtonHandler will be invoked

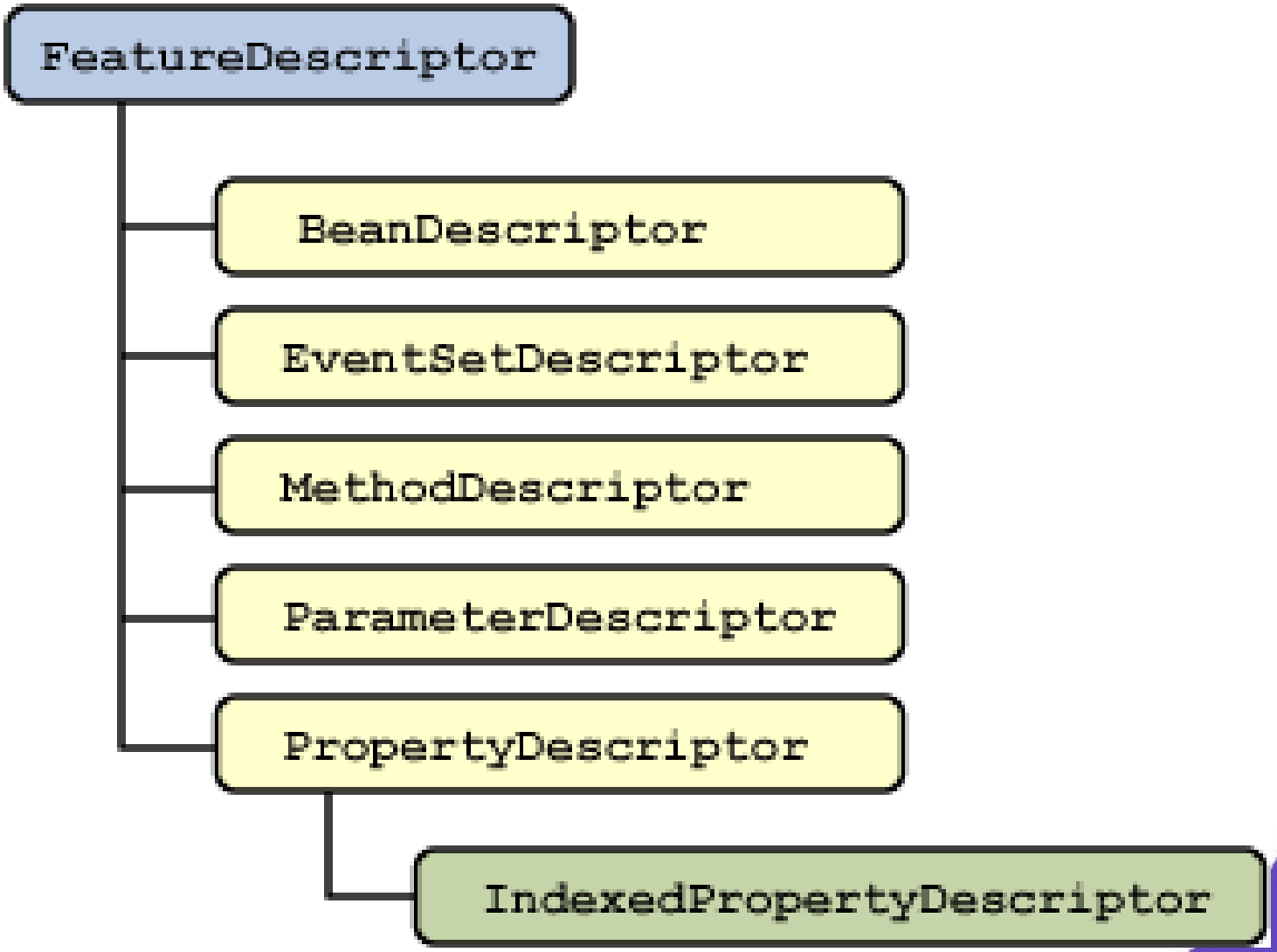


Introspection

What is Introspection?

- Introspection is the automatic 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 set/getProperty() and add/removeListener()

FeatureDescriptor

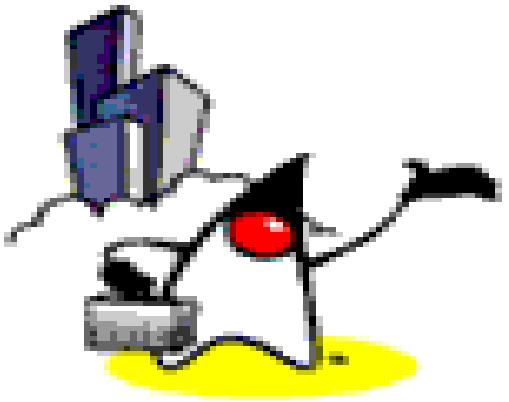


Things That Can be Found through Introspection

- 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

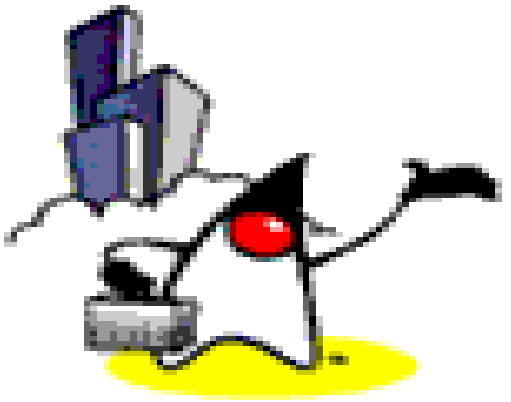
- Multicast events
  - public void addEventListenerType(EventListenerType l);
  - public void removeEventListenerType(EventListenerType l);
- Unicast events
  - public void addEventListenerType(EventListenerType l) throws TooManyListenersException;
  - public void removeEventListenerType(EventListenerType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



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 on a Microsoft Windows machine, the serialized file can be sent with e-mail to a Solaris machine, and then a Java application can restore the Frame window to the exact state which existed on the Microsoft Windows machine.



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>
```

JavaBeans