JavaBeans Component Development

SL-291
Course Contents

### About This Course

- Course Goal
- Course Overview
- Course Map
- Module-by-Module Overview
- Course Objectives
- Skills Gained by Module
- Guidelines for Module Pacing
- Topics Not Covered
- How Prepared Are You?
- Introductions
- How to Use Course Materials
- How to Use the Icons
- Typographical Conventions and Symbols

### Overview of JavaBeans

- Module Overview
- What Is JavaBeans?
- What Is a Bean?
- Design Goals of JavaBeans
- Component Architectures
- Putting Beans Together
- Life Cycle of a Bean
- The BDK 1.0 Overview
- ActiveX as a Component Model
Persistence ................................................................. 7-1
  Module Overview ......................................................... 7-2
  Goals for Bean Storage .................................................. 7-3
  Java Object Serialization ............................................. 7-4
  What Is and Is Not Saved ............................................... 7-5
  Input and Output Interfaces ........................................... 7-6
  Saving Beans to Streams ............................................... 7-7
  Retrieving Beans From Streams ..................................... 7-8
  defaultReadObject and defaultWriteObject Methods ........ 7-9
  Sample Code .............................................................. 7-11
  Sample Code Discussion ............................................... 7-15
  Deserialization and Bean instantiation ............................. 7-16
  Creating a Java Beans Prototype ..................................... 7-17
  Packaging a Prototype Bean .......................................... 7-18
  Recap of Persistence .................................................... 7-19
  Exercise: Creating a New Bean Through Serialization ........ 7-20
  Check Your Progress .................................................... 7-21
  Think Beyond ............................................................. 7-22
Implementing a Customizer Class ................................................................. 9-5
Defining BeanInfo ......................................................................................... 9-6
Extending Component/Implementing Customizer ........................................... 9-7
Adding and Removing PropertyChangeListeners ........................................ 9-8
Defining setObject() .................................................................................... 9-9
Example of a Customizer ............................................................................... 9-10
Recap of Customizers .................................................................................... 9-12
Exercise: Creating a Customizer ................................................................. 9-13
Check Your Progress ..................................................................................... 9-14
Think Beyond ............................................................................................... 9-15

Event Adapters .............................................................................................. 10-1
Module Overview ............................................................................................ 10-2
What Is an Event Adapter? ............................................................................. 10-3
Adapters Used in the BeanBox ....................................................................... 10-4
Types of Adapters .......................................................................................... 10-5
Adapter Diagrams .......................................................................................... 10-6
Differentiating Adapters From Normal Listeners ........................................... 10-7
Demultiplexing Adapter Example ................................................................. 10-8
Description of Application ............................................................................ 10-9
Builder.java Code ........................................................................................ 10-10
Widgets.java Code ......................................................................................... 10-11
ActionAdapter.java Code .............................................................................. 10-13
WhatToDo.java Code ..................................................................................... 10-15
Multiplexing Adapters .................................................................................. 10-16
Overview of Multiplexer Exercise .................................................................. 10-17
Exercise: Working With Adapters ............................................................... 10-18
Check Your Progress ..................................................................................... 10-19
Think Beyond ............................................................................................... 10-20
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handling or Sharing Data Among Beans</td>
<td>13-9</td>
</tr>
<tr>
<td>InfoBus Technology</td>
<td>13-10</td>
</tr>
<tr>
<td>Overview of the InfoBus Architecture</td>
<td>13-11</td>
</tr>
<tr>
<td>InfoBus Classes and Interfaces</td>
<td>13-12</td>
</tr>
<tr>
<td>Code Samples From the InfoBus Software</td>
<td>13-14</td>
</tr>
<tr>
<td>InfoBus Events</td>
<td>13-15</td>
</tr>
<tr>
<td>InfoBus Event Listeners</td>
<td>13-16</td>
</tr>
<tr>
<td>Events, Firing Methods, and Handlers</td>
<td>13-17</td>
</tr>
<tr>
<td>InfoBus DataItem Interface</td>
<td>13-18</td>
</tr>
<tr>
<td>Sample DataItems From the InfoBus Software</td>
<td>13-19</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>13-20</td>
</tr>
<tr>
<td>javaxbeans.beancontext Package</td>
<td>13-21</td>
</tr>
<tr>
<td>Beans and BeanContexts</td>
<td>13-22</td>
</tr>
<tr>
<td>BeanContext Interface</td>
<td>13-23</td>
</tr>
<tr>
<td>BeanContextServices Interface</td>
<td>13-24</td>
</tr>
<tr>
<td>Providing a Bean With a BeanContext</td>
<td>13-25</td>
</tr>
<tr>
<td>BeanContext Support for Applets</td>
<td>13-26</td>
</tr>
<tr>
<td>BeanContext Services Support</td>
<td>13-27</td>
</tr>
<tr>
<td>BeanContext Support Classes</td>
<td>13-28</td>
</tr>
<tr>
<td>javaxbeans Activation Framework</td>
<td>13-29</td>
</tr>
<tr>
<td>Major Elements Comprising the JAF Architecture</td>
<td>13-30</td>
</tr>
<tr>
<td>Overview of the Major Elements</td>
<td>13-31</td>
</tr>
<tr>
<td>Check Your Progress</td>
<td>13-33</td>
</tr>
<tr>
<td>Think Beyond</td>
<td>13-34</td>
</tr>
</tbody>
</table>
Preface

About This Course
Course Goal

This course provides you with knowledge and skills to

- Create reusable bean components
- Create bean properties
- Understand how introspection and reflection works
- Work with the bean event model
- Customize and persist beans
Course Overview

- Use bean components to create new applications
- Create beans using conventions in the JavaBeans™ API specification
- Use beans to bridge to component models that do not support Java™ technology
- See how beans can run in any environment that supports Java technology
Module-by-Module Overview

- Module 1 – “Overview of JavaBeans”
- Module 2 – “The BeanBox”
- Module 3 – “The Bean Event Model”
- Module 4 – “Bean Conventions”
- Module 5 – “Bean Properties”
- Module 6 – “Introspection”
- Module 7 – “Persistence”
Module-by-Module Overview

• Module 8 – “Property Sheets and Property Editors”
• Module 9 – “Customizers”
• Module 10 – “Event Adapters”
• Module 11 – “Distributed Computing With Beans”
• Module 12 – “Beans Outside of the BeanBox”
• Module 13 – “Business Environment for JavaBeans”
Course Objectives

• Define a bean component and describe why JavaBeans is a Java component model
• Package JavaBeans components into JAR files, add them to the BeanBox tool palette, and test them in the BeanBox
• Given a class that implements a specific listener interface, write the appropriate event handling methods
• Create a JavaBeans component with bound or constrained properties
• Describe how the introspection process works, including the relevance to naming conventions and to menu options displayed in the Beanbox
• Write the required persistence mechanisms for a customized bean component
• Control the configuration and customization of bean components through customizer classes, property editors, property sheets, and BeanInfo classes
• Create event adapters to modify event delivery between sources and listeners
• Develop bean components as intelligent front-ends to network servers using a network access mechanism (such as JDBC, RMI, or CORBA)
• Create applets or applications using existing bean components
• Explain how JavaBeans components can be used with existing component models such as ActiveX
**Skills Gained by Module**

**Meaning of:**

- Black boxes
- Gray boxes

<table>
<thead>
<tr>
<th>Skills Gained</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill or Objective 1</td>
<td>1</td>
</tr>
<tr>
<td>Skill or Objective 2</td>
<td>2</td>
</tr>
<tr>
<td>Skill or Objective 3</td>
<td>3</td>
</tr>
<tr>
<td>Skill or Objective 4</td>
<td>4</td>
</tr>
</tbody>
</table>
## Guidelines for Module Pacing

<table>
<thead>
<tr>
<th>Module</th>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
<th>Day 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>“About This Course”</td>
<td>A.M.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 1 – “Overview of JavaBeans”</td>
<td>A.M.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 2 – “The BeanBox”</td>
<td>A.M./P.M.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 3 – “The Bean Event Model”</td>
<td>P.M.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module 4 – “Bean Conventions”</td>
<td></td>
<td></td>
<td>A.M.</td>
<td></td>
</tr>
<tr>
<td>Module 5 – “Bean Properties”</td>
<td></td>
<td></td>
<td>A.M.</td>
<td></td>
</tr>
<tr>
<td>Module 6 – “Introspection”</td>
<td></td>
<td></td>
<td>P.M.</td>
<td></td>
</tr>
<tr>
<td>Module 7 – “Persistence”</td>
<td></td>
<td></td>
<td>P.M.</td>
<td></td>
</tr>
<tr>
<td>Module 8 – “Property Sheets and Property Editors”</td>
<td></td>
<td></td>
<td>A.M.</td>
<td></td>
</tr>
<tr>
<td>Module 9 – “Customizers”</td>
<td></td>
<td></td>
<td>A.M./P.M.</td>
<td></td>
</tr>
<tr>
<td>Module 10 – “Event Adapters”</td>
<td></td>
<td></td>
<td>P.M.</td>
<td></td>
</tr>
<tr>
<td>Module 11 – “Distributed Computing With Beans”</td>
<td></td>
<td></td>
<td></td>
<td>A.M.</td>
</tr>
<tr>
<td>Module 12 – “Beans Outside of the BeanBox”</td>
<td></td>
<td></td>
<td>A.M./P.M.</td>
<td></td>
</tr>
<tr>
<td>Module 13 – “Business Environment for JavaBeans”</td>
<td></td>
<td></td>
<td>P.M.</td>
<td></td>
</tr>
</tbody>
</table>
Topics Not Covered

- Object-oriented concepts
- Object-oriented design and analysis
- Java language constructs
- Details on distributed programming APIs
How Prepared Are You?

• Experienced programmer able to use AWT components, layout managers, event handling in the Java programming language?

• Able to implement interfaces and exception handling?

• Experienced with object-oriented programming languages?

• Capable of designing an object-oriented model for a problem?

• At ease with learning new APIs?

• Able to learn from code examples and a technical explanation?
Introductions

• Name
• Company affiliation
• Title, function, and job responsibility
• Distributed computing experience
• Component development experience
• Application builder tool experience
• Reasons for enrolling in this course
• Expectations for this course
How to Use Course Materials

• Relevance
• Overhead image
• Lecture
• Exercise
• Check your progress
• Think beyond
How to Use the Icons

• Demonstration

• Discussion

• Exercise

• Additional resources
Typographical Conventions and Symbols

- Courier is used for the names of commands, files, directories, and parts of the Java programming language, as well as on-screen computer output.

- **Courier bold** is used for characters and numbers that you type.

- **Courier italic** is used for variables and command-line placeholders that are replaced with a real name or value.

- *Palatino italics* is used for book titles, new words or terms, or words that are emphasized.
Module 1

Overview of JavaBeans
Module Overview

- Objectives
- Relevance
- References
What Is JavaBeans?

- A Java component model
- JavaBeans APIs
- Extension of Java platform
What Is a Bean?

- Definition
- Features of beans
- Examples of beans
- Classes and beans
Design Goals of JavaBeans

- Compact: Leverage the strengths of the Java platform
- Easy to create and use
- Fully portable
- Flexible build-time component editors
- Leverage distributed computing mechanisms
Component Architectures

• Why are they useful?
• Services of component models
  • Component interface publishing and discovery
  • Event handling
  • Persistence
  • Layout control
  • Application builder support
Putting Beans Together

Describe how the following beans might be hooked together:

- Graphing or charting bean
- Random number generator bean
- Animation bean
- Timer bean
Life Cycle of a Bean

- Development
- Design time
- Run time
The BDK 1.0 Overview

- The BDK provides a reference implementation of the JavaBeans Specification and is intended for bean developers and tool vendors.

- The BDK contains
  - A reference bean container called the BeanBox
  - Sample source code for developing JavaBeans components
  - An on line tutorial for developing JavaBeans

- Use the BDK with JDK™ 1.1.5 or later.
The BDK 1.0 Overview

Hashed boundary around the BeanBox bean indicates the currently selected bean.
ActiveX as a Component Model

• Overview

• Advantages of ActiveX
JavaBeans and ActiveX Comparison

- Platform
- Heavyweight or lightweight
- Network device support
- Interoperability
- Software versioning and distribution
- Distributed computing
- Performance
- Security
Check Your Progress

• Define JavaBeans
• Explain what a bean is
• Describe the design goals for JavaBeans
• Explain why JavaBeans is a Java component model
• Describe the services every component model must provide
• Compare and contrast JavaBeans and ActiveX as component models
Think Beyond

Suppose you have written a few JavaBeans components.

You now want to test how well your beans work.

Can you do this within the BDK? Do you need a third-party builder tool?
Module 2

The BeanBox
Module Overview

• Objectives
• Relevance
• References
What Is the BeanBox?

- Definition
- Running the BeanBox
Windows of the BeanBox

- Toolbox
  - BlueButton
  - OrangeButton
  - OurButton
  - ExplicitButton
  - EventMonitor
  - JellyBean
  - Juggler
  - TickTock
  - Voter
  - ChangeReporter
  - Molecule
  - QuoteMonitor
  - JDBC SELECT
  - SorterBean
  - BridgeTester
  - Transitional Bean

- Properties - BeanBox
  - background
  - foreground
  - name: panel1
  - font: Abode...

Hashed boundary around the BeanBox bean
Windows of the BeanBox

- ToolBox window
- BeanBox window
- Properties window
Design Mode in the BeanBox

- When started, the BeanBox is in design mode
- To disable design mode, select View ➤ Disable Design Mode
- What can you do in design mode
- What can you do in run-time mode
Manipulating a Bean

• Placing a bean on the BeanBox window
• Selecting a bean
• Moving a bean
• Resizing a bean
Changing Bean Properties – Properties Window

- Mechanics
- Effects on the BeanBox window
Changing Bean Properties – Customizers

• Customizer class
• Edit ➤ Customize option
• Example beans with customizers
Bound Properties

- Source bean with the bound property
- Target bean and target property
- Mechanics of connecting them
  - Selecting Edit ➤ Bind property
  - Selecting source property
  - Selecting target bean and target property
- What happens after the beans are connected
Connecting Beans With Event Handlers

1. Select the source bean.
2. Select the event using Edit ➤ Events submenu.
3. Select the target bean.
4. Select the handler method.
Saving and Restoring the BeanBox

- Use the File ➤ Save option.
- What is saved?
- What type of file is created?
Adding Beans to the ToolBox Window

- JAR files
- Manifest files
- Specifying a JAR file in an HTML file
- Creating JAR files
Home Directory Structure

```
Labfiles
  ├── BDK
  │   ├── jars
  │   │   └── sunw
  │   └── demo
  │       ├── beanbox
  │           └── run
  │           └── command
  │                   └── and so on
  │   └── demo
  │       └── bdk
  │           └── sunw
  │                   └── and so on
  └── packages
      ├── provided files
      │   └── manifest
      ├── sesbeans
      └── beanbox
          └── ... outsideBB
              └── and so on
      └── examples
          └── solutions
              └── get_started
```

Check Your Progress

- Explain the purpose of the BeanBox
- Move and resize beans on the Composition window of the BeanBox
- Change bean properties using the Properties window of the BeanBox
- Register a bean as the listener of an event generated by another bean in the Composition window
- Save and restore the current state of the BeanBox
- Explain what a Java archive (JAR) file is and how it can be used
- Create a JAR file for a prewritten bean and add it to the ToolBox window of the BeanBox
Think Beyond

• How does the event handling work for beans?
• How do you define your own events?
• How do you indicate that you want to receive a particular event?
Module 3

Bean Event Model
Module Overview

- Objectives
- Relevance
- References
What Is an Event?

- Definition
- Examples
  - Window events
  - Mouse events
  - Keyboard events
  - List events
  - Scrolling events
Delegation Model Overview

- Sources and listeners
- Propagating notification of events
import java.awt.*;
import java.awt.event.*;

public class ButtonHandler implements ActionListener {
    /**
     * Component that will contain messages about
     * events generated.
     */
    TextArea output;

    /**
     * Creates an ActionListener that will put messages in
     * TextArea everytime event received.
     */
    public ButtonHandler(TextArea c) {
        output = c;
    }

    /**
     * When receives action event notification, appends
     * message to the TextArea passed into the constructor.
     */
    public void actionPerformed(ActionEvent e) {
        output.append("Action occurred:" + e + "]");
    }
}

class ActionTester {
    public static void main(String args[]) {
        Frame f = new Frame("Button Handler");
        TextArea area = new TextArea(6, 80);
        Button button = new Button("Fire Event");
        button.addActionListener(new ButtonHandler(area));
        f.add(button, BorderLayout.NORTH);
        f.add(area, BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
    }
}
Code Explanation

Key items in the code:

- Registering listeners for the event using addActionListener()

- Implementing the ActionListener interface

- Defining the required ActionListener event handler, actionPerformed()
Categories of Events

- Category – XXX
  - Action, Item, Mouse motion, Mouse button, Key, Focus, Adjustment, Component, Window, Container, and Text

- Interface – XXXListener
  - ActionListener, ItemListener, and so on

- Event – XXXEvent
  - ActionEvent, ItemEvent, MouseEvent, and so on
Obtaining Details About the Event

- All events have `java.util.EventObject` as a base class.

- Events have accessor methods.
  - For example, `getSource()` gets the object that generated the event.

- You should check event classes in the `java.awt.event` package for examples of events generated by components in the AWT.
Creating Your Own Event

• Extend `java.util.EventObject` or an AWT event class

• Define any accessor methods for listeners to obtain information about the event

• Example

```java
package sesbeans.stock;
import java.util.*;
public class StockPriceChangeEvent extends EventObject {
    private Stock stock;

    public StockPriceChangeEvent (Object source, Stock s) {
        super(source);
        stock = s;
    }

    public Stock getStock() {
        return stock;
    }
}
```
Listeners

- Identify listeners
- Listener interfaces
  - ActionListener interface

```java
package java.awt.event;
import java.util.EventListener;

/**
 * The listener interface for receiving action events.
 */
public interface ActionListener extends EventListener {
    /**
     * Invoked when an action occurs.
     */
    public void actionPerformed(ActionEvent e);
}
```
Creating Your Own Listener Interface

- Extend `java.util.EventListener`
- Specify the handler method, with the event type as an argument
- Example

```java
package sesbeans.stock;

import java.util.EventListener;

public interface StockPriceChangeListener extends EventListener {
    public void priceChange(StockPriceChangeEvent e);
}
```
Event Sources

• Using common sources
• Creating your own source
• Identifying sources
  • addXXXListener
  • removeXXXListener
Multicast Syntax

```java
private Vector listeners = new Vector();

public void addStockPriceChangeListener(StockPriceChangeListener spcl) {
    listeners.addElement(spcl);
}

public void removeStockPriceChangeListener(StockPriceChangeListener spcl) {
    listeners.removeElement(spcl);
}
```
private StockPriceChangeListener listener = null;

public void addStockPriceChangeListener(StockPriceChangeListener spcl) throws java.util.TooManyListenersException {
    if (listener == null) {
        listener = spcl;
    } else {
        throw new java.util.TooManyListenersException();
    }
}

public void removeStockPriceChangeListener(StockPriceChangeListener spcl) {
    if (listener == spcl) {
        listener = null;
    }
}
private void generateStockEvent() {
    StockPriceChangeEvent event;
    event = new StockPriceChangeEvent(this, stock);

    Vector lis = (Vector)listeners.clone();
    StockPriceChangeListener spcl;
    for (int i=0, len=lis.size(); i<len; i++) {
        spcl = (StockPriceChangeListener)lis.elementAt(i);
        spcl.priceChange(event);
    }
}
Event Delivery Issues

• Synchronous delivery
• Multiple listeners
Recap of Event Model

- XXXListener interface
- XXXEvent
- Event source
- Event listener
Bean Components and Event Handling

- Beans are connected in BeanBox using Edit ➤ Events.
- In the example of stock market beans:
  - StockWatcher bean generates an event when the price of stock changes.
  - StockDetail bean describes a stock and receives price change notifications.
- StockWatcher bean is nonvisual.
- The price of a stock can change every 5 seconds.
package sesbeans.stock;

public class Stock {
    private String company;
    private String symbol;
    private double price;

    public Stock(String co, String sym, double p) {
        company = co;
        symbol = sym;
        price = p;
    }

    public void setCompany(String co) {
        company = co;
    }

    public String getCompany() {
        return company;
    }

    public void setSymbol(String sym) {
        symbol = sym;
    }

    public String getSymbol() {
        return symbol;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double p) {
        price = p;
    }
}
StockPriceChangeEvent

Code

```java
package sesbeans.stock;

import java.util.*;

public class StockPriceChangeEvent extends EventObject {
    private Stock stock;

    public StockPriceChangeEvent(Object source, Stock s) {
        super(source);
        stock = s;
    }

    public Stock getStock() {
        return stock;
    }
}
```

```java
package sesbeans.stock;

import java.util.EventListener;

public interface StockPriceChangeListener extends EventListener {
    public void priceChange(StockPriceChangeEvent e);
}
```
package sesbeans.stock;

import java.awt.event.*;
import java.util.Vector;
import sesbeans.beans.*;

public class StockWatcher implements ActionListener {
    Stock stock;
    Vector listeners = new Vector();

    public StockWatcher() {
        this(new Stock("Sun Microsystems", "SUNW", 50));
    }

    public StockWatcher(Stock s) {
        stock = s;

        //create a timer that generates events every 5 secs,
        //register this class as interested in timer ticks, and
        //make timer active.
        TimerBean t = new TimerBean(5000);
        t.addActionListener(this);
        t.setActive(true);
    }

    public String getCompany() {
        return stock.getCompany();
    }

    public void setCompany(String co) {
        stock.setCompany(co);
    }
}
```java
33
34    public String getSymbol() {
35        return stock.getSymbol();
36    }
37
38    public void setSymbol(String s) {
39        stock.setSymbol(s);
40    }
41
42    public void setPrice(double p) {
43        stock.setPrice(p);
44    }
45
46    public double getPrice() {
47        return stock.getPrice();
48    }
49
50    public void actionPerformed(ActionEvent e) {
51        double random = Math.random();
52
53        if (random < .28) {
54            //price went down
55            stock.setPrice(stock.getPrice()-.25);
56        } else if (random > .7) {
57            //price went up
58            stock.setPrice(stock.getPrice()+.25);
59        } else {
60            //do not generate event b/c price did not change
61            return;
62        }
63
64        generateStockEvent();
65    }
66
```
67    public void addStockPriceChangeListener(
68      StockPriceChangeListener spl) {
69        listeners.addElement(spl);
70    }
71
72    public void removeStockPriceChangeListener(
73      StockPriceChangeListener spl) {
74        listeners.removeElement(spl);
75    }
76
77    private void generateStockEvent() {
78        StockPriceChangeEvent event;
79        event = new StockPriceChangeEvent(this, stock);
80
81        Vector lis = (Vector)listeners.clone();
82        StockPriceChangeListener spcl;
83        for (int i=0, len=lis.size(); i<len; i++) {
84            spcl = (StockPriceChangeListener)lis.elementAt(i);
85            spcl.priceChange(event);
86        }
87    }
88  }
package sesbeans.stock;

import java.awt.TextArea;

public class StockDetail extends TextArea implements StockPriceChangeListener {
    public StockDetail() {
        this(null);
    }

    public StockDetail(Stock s) {
        super(4, 30);
        showDetail(s);
    }

    public void priceChange(StockPriceChangeEvent e) {
        showDetail(e.getStock());
    }

    public void showDetail(Stock s) {
        if (s != null) {
            setText("Company: "+ s.getCompany() + "\n");
            append("symbol: " + s.getSymbol() + "\n");
            append("price: " + s.getPrice());
        } else {
            setText(""");
        }
    }
}
Running Stock Market Beans

- Create a JAR file and load it in BeanBox
- Create instances of StockWatcher and StockDetail beans
- Connect StockWatcher to StockDetail using the StockPriceChangeEvent
Exercise: Working With the Bean Event Model

- Objective
- Preparation
- Tasks
- Exercise summary
Check Your Progress

• Define event, event source, and event listener

• Define the difference between multicast and unicast sources

• Create a multicast or unicast source

• Implement a specified listener interface

• Create two simple bean components in which one is a listener for the events of the other
Think Beyond

In working with the BeanBox, different properties were displayed in the PropertySheet window for each bean you selected on the Composition window.

• How do you create bean properties?

• Are there different types of properties?
Module 4

Bean Conventions
Module Overview

• Objectives
• Relevance
• References
Introduction to Introspection

• What problem does introspection solve?
  • The code integration problem
  • Introspection and JavaBeans API
Introspection Addresses Key Issues

- Reuse affected by different coding styles used by developers
- Event propagation model before JDK 1.1
- Lack of support for examination and invocation of methods before to JDK 1.1
Definitions

• Reflection
• Naming conventions
• Introspection
• BeanInfo
Sample Uses for BeanInfo

- Limit a long list
- Provide GIF images as an icon
- Add a descriptive name for the properties
- Affect advanced options
- Specify additional "smart" customizer classes
The Introspector

• What the Introspector is

• How it follows a plan for filling out Descriptor classes

  • Finds information using `BeanInfo` classes and `getBeanInfo()`

  • Uses Reflection API classes
Naming Conventions for Properties

• Simple properties
  • public PropertyType getProperty
  • public void setProperty(PropertyType a)

• Boolean properties
  • public boolean isProperty()
  • public void setProperty(boolean b)
Naming Conventions for Properties

- Indexed properties

```java
public PropertyElement getProperty_Name(int index)
public void setProperty_Name(int index, PropertyElement element)

public PropertyElement[] getProperty_Name()
public void setProperty_Name(PropertyElement element[])
```
Naming Conventions for Events

• Multicast event sources

```java
public void addEventNameListner(EventNameListner el)
```

```java
public void removeEventNameListner(EventNameListner el)
```
Naming Conventions for Events

• Unicast event sources

```java
public void addEventNameListener(EventNameListener el)
        throws java.util.TooManyListenersException

public void removeEventNameListener(EventNameListener el)
```
Naming Conventions for Methods

- Accessibility and public methods
  - Properties
  - Events
- Capitalization rules
Check Your Progress

- Define introspection and reflection
- Analyze the relationship between introspection and the naming conventions used for properties, events, and methods
Think Beyond

In working with the BeanBox, you might have noticed menu choices on the Edit menu that referred to Bound or Constrained properties.

What are these exactly, how do they work, and how do you create them?
Module 5

Bean Properties
Module Overview

• Objectives
• Relevance
• References
What Is a Bean Property?

- Definition

- Types of bean properties
  - Simple
  - Bound
  - Constrained
  - Indexed
Simple Properties

- Defining simple properties
- Adding simple properties to a bean
Adding Simple Properties

1 package sesbeans.circle;
2
3 import java.awt.*;
4 import java.beans.*;
5
6 /** A simple Bean that is a Circle with properties to
7 * change color, change radius, calculate circumference
8 */
9
10 public class CircleBean extends Canvas {
11    private Color color = Color.blue;
12    private int radius;
13    private double circumference;
14
15    //Construct a small circle
16    public CircleBean() {
17        setSize(new Dimension(60,60));
18        this.radius = 100;
19        this.circumference = getCircum();
20    }
21
22    public void paint(Graphics g) {
23        g.setColor(color);
24        g.fillArc(0,0,radius,radius,0,360);
25    }
26
27    // Read-write properties - these show up in the BeanBox
28    public Color getColor() {
29        return color;
30    }
31
32    public void setColor(Color newColor) {
33        color = newColor;
34        repaint();
35    }
```java
36  public int getRadius() {
37     return radius;
38  }
39 
40 
41  public void setRadius(int r) {
42     radius = r;
43     circumference = 2 * 3.14159 * radius;
44     System.out.println("Circumference: " + circumference);
45     repaint();
46  }
47 
48  // read-only property - does not show up in BeanBox
49  public double getCircum() {
50     return circumference;
51  }
52 }
```
Properties and Edit Menu

radius property

radius

color property

color

background

foreground

name

font

File | Edit | View
---|---|---
Cut | Copy | Paste
Report... | Events
Bound and Constrained Properties

• In addition to simple, boolean, and indexed property types, properties can also be bound or constrained.

• Support classes are provided in the JavaBeans API for creating bound and constrained properties.
Bound Properties

• Definition

• Defining bound properties

  ![Diagram](image)

  - Property A is a bound property
  - Bean A creates a `PropertyChangeSupport` object
  - Bean A fires the `PropertyChangeEvent` in its `setX` method
  - Choose a target property (with same property type as property X):
    - `Property Bx`
    - `Property By`
    - `Property Bz`

• `PropertyChangeSupport` class

• Modifying the property `set` method
  - *Fire the `PropertyChangeEvent`*
Example of Creating a Bound Property

```java
15    private PropertyChangeSupport support = new PropertyChangeSupport(this);
16
18    //registration methods for PropertyChangeListener
19    public void addPropertyChangeListener(PropertyChangeListener pcl) {
20        support.addPropertyChangeListener(pcl);
21    }
22
23    public void removePropertyChangeListener(PropertyChangeListener pcl) {
24        support.removePropertyChangeListener(pcl);
25    }

43    public void setColor(Color newColor) {
44        Color prevColor = color;
45        color = newColor;
46        support.firePropertyChange("color", prevColor, color);
47        repaint();
48    }
```
Bound Properties and the BeanBox

Edit menu change

[Diagram showing the Edit menu with the 'Bind property...' option selected, and a Property Name Dialog window with a list of properties: radius, color, background, foreground, circum, font.]
Recap of Bound Properties

Code Pieces for Defining Bound Property abc

```java
private ABCtype abc;
private PropertyChangeSupport support = new PropertyChangeSupport(this);

//registration methods for PropertyChangeListener
public void addPropertyChangeListener(PropertyChangeListener pl) {
    support.addPropertyChangeListener(pl);
}

public void removePropertyChangeListener(PropertyChangeListener pl) {
    support.removePropertyChangeListener(pl);
}

public ABCtype getABC() {
    return abc;
}

public void setABC(ABCtype newABC) {
    ABCtype oldABC = abc;
    abc = newABC;
    support.firePropertyChange("abc", oldABC, newABC);
}
```

<table>
<thead>
<tr>
<th>Listener Interface</th>
<th>PropertyChangeListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>PropertyChangeEvent</td>
</tr>
<tr>
<td>Handler method</td>
<td>propertyChange</td>
</tr>
<tr>
<td>Utility class</td>
<td>PropertyChangeSupport</td>
</tr>
</tbody>
</table>
Constrained Properties

• Definition

• Overview
  • Order tasks are done in setXXX
Constrained Properties

• Defining constrained properties
  • `set` method throws `PropertyVetoException`

  ![Diagram]

  Property Y is a constrained property
  Bean A creates a `VetoableChangeSupport` object
  Bean A defines the `setY` method to throw a `PropertyVetoException`

• Handling vetoes

• `VetoableChangeListener`
Constrained Properties

• Using the VetoableChangeSupport class
  • Utility class is similar to PropertyChangeSupport
  • What the utility class does for you

• Registering listeners

• Modifying the property set method
  • Calls the fireVetoableChange() method of the VetoableChangeSupport object
Example of Creating a Constrained Property

```java
private VetoableChangeSupport vetos = new VetoableChangeSupport(this);

public void setRadius(int r) throws PropertyVetoException {
    int prevRadius = radius;
    // check for negative values
    if (r < 0) {
        System.out.println("Negative radius not allowed!");
    } else {
        vetos.fireVetoableChange("radius",
                        prevRadius, r);
        // no one vetoed, so make the change
        radius = r;
        circumference = 2 * 3.14159 * radius;
        System.out.println("Circumference " + circumference);
        repaint();
    }
}
```
Constrained Properties and Validation

• Using a source or listener
• Validating the property change
  • Listener implements VetoableChangeListener
  • Listener defines vetoableChange
  • vetoableChange validates proposed property value
  • vetoableChange throws PropertyVetoException if the value is unacceptable
Constrained Properties and the BeanBox

• VetoableChange added to Edit ➤ Events submenu
• No special Edit option as for bound properties
Recap of Constrained Properties

Code Pieces for Defining Constrained Property xyz

```java
private XYZtype xyz;
private VetoableChangeListener vetoes = new VetoableChangeListener(this);

//registration methods for VetoableChangeListeners
public void addVetoableChangeListener(VetoableChangeListener vl) {
    vetoes.addVetoableChangeListener(vl);
}

public void removeVetoableChangeListener(VetoableChangeListener vl) {
    vetoes.removeVetoableChangeListener(vl);
}

public XYZtype getXYZ() {
    return abc;
}
public void setXYZ(XYZtype newXYZ) throws PropertyVetoException{
    XYZtype oldXYZ = xyz;
    vetoes.fireVetoableChange("xyz", oldXYZ, newXYZ);
    xyz = newXYZ;
}
```

<table>
<thead>
<tr>
<th>Listener interface</th>
<th>VetoableChangeListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>PropertyChangeEvent</td>
</tr>
<tr>
<td>Handler method</td>
<td>vetoableChange(PropertyChangeEvent evt) throws PropertyVetoException</td>
</tr>
<tr>
<td>Utility class</td>
<td>VetoableChangeListener</td>
</tr>
</tbody>
</table>

Listeners define vetoableChange(PropertyChangeEvent evt)
throws PropertyVetoException { ... }
The method will validate the proposed new value for property xyz and throw PropertyVetoException if the value is not allowed.
Properties and the BeanBox

- Discovering bean properties
  - Only read-write properties are displayed on the PropertySheet window

- Properties on the Properties window
  - Property editors
  - Editors provided with JavaBeans
Views on the Properties Window

- Different views
  - PropertyCanvas
  - PropertySelector
  - PropertyText
- How a view is determined
Example of Views

Properties – Molecule
- moleculeName
  - HyaluronicAcid
  - benzene
  - buckminsterfullerine
  - cyclohexane
  - ethane
  - water

Properties – OurButton
- fontSize
  - 1 1/2
- backgroundColor
- debug
  - true
- largeFont
  - false
- foreground
  - black
- label
  - press
- font

PropertySelector views
- PropertyCanvas views
- PropertyText views

Dialog window
Exercise: Defining Bean Properties

- Objective
- Preparation
- Tasks
- Exercise summary
Check Your Progress

• Define a bean property
• Compare the different types of bean properties
• Explain how the naming conventions for `get` and `set` methods are used to locate properties
• Determine if a property is read-write, read-only, or write-only
• Create a bean component with a bound property
• Create a bean component with a constrained property
Think Beyond

- What is introspection?

- How does it work to discover the properties and behaviors of a bean?
Module 6

Introspection
Module Overview

- Objectives
- Relevance
- References
Advantages Provided by Introspection

• Foundation concept that enables the JavaBeans architecture to be effective
  • Portability
  • Reuse

• Introspection, reflection, and BeanInfo classes
Bean Creation and Analysis

- **Beans**
  - **create a Bean**
    - `instantiate()`
  - **Used for introspection**
    - `getBeanInfo()`

- **Introspector**
  - **interface**
    - `BeanInfo`

- **Used by developers**
  - `SimpleBeanInfo`
  - `GenericBeanInfo`

- **reflect**
  - **Used for reflection**
    - `various classes`
Beans\_instantiate Method

The builder tool instantiates a bean using

```java
String beanName = "beanco.chart.PieChart";
Component bean = (Component) Beans.instantiate(classLoader, beanName);
```
Instantiation Supports Customized Beans and Applets

- The second argument to `Beans.instantiate` is a String name for a bean. The bean can be a
  - Serialized file
  - Class file
  - Applet
**Introspector.getBeanInfo** Method

- Called after **Beans.instantiate()**
- Use code similar to

```java
Class beanClass = Class.forName("beanco.chart.PieChart");
BeanInfo bi = Introspector.getBeanInfo(beanClass);
```
Information Discovered by 
getBeanInfo

CLASS: sesbeans.dataTable.DataTableBean

Properties:

rows int getRows/setRows
getCellSize/setCellSize
foreground class java.awt.Color getForeground/setForeground
background class java.awt.Color getBackground/setBackground
font class java.awt.Font getFont/setFont
name class java.lang.String getName/setName
columns int getColumns/setColumns
constrained boolean getConstrained/setConstrained
...

Event sets:

vetoableChange addVetoableChangeListener/removeVetoableChangeListener
vetoableChange

mouse addMouseListener/removeMouseListener
mouseClicked
mousePressed
mouseReleased
mouseEntered
mouseExited
Information Discovered by getBeanInfo

Methods:
public void sesbeans.dataTable.DataTableBean.
removeVetoableChangeListener(java.beans.VetoableChangeListener)
    public synchronized void sesbeans.dataTable.DataTableBean.setRows
        (int) throws java.beans.PropertyVetoException
        public boolean sesbeans.dataTable.DataTableBean.getConstrained()
        public void sesbeans.dataTable.DataTableBean.
addVetoableChangeListener(java.beans.VetoableChangeListener)
    public int sesbeans.dataTable.DataTableBean.getCellSize()
        public void sesbeans.dataTable.DataTableBean.
handleDataTableEvent(sesbeans.dataTable.DataTableBean)
    public synchronized void sesbeans.dataTable.DataTableBean.
setCellSize(int)
        public synchronized void sesbeans.dataTable.DataTableBean.setColumns(int)
        throws java.beans.PropertyVetoException
        public synchronized void sesbeans.dataTable.DataTableBean.
setConstrained(boolean)
    public int sesbeans.dataTable.DataTableBean.getColumns()
        public int sesbeans.dataTable.DataTableBean.getRows()
        ...

...
SimpleBeanInfo Class

- Implements the BeanInfo interface
- Defines methods specified by the interface (return null)
  - getBeanDescriptor()
  - getAdditionalBeanInfo()
  - getPropertyDescriptors()
  - getDefaultPropertyIndex()
  - getEventSetDescriptors()
  - getDefaultEventIndex()
  - getMethodDescriptors()
  - getIcon(int)
A BeanInfo Class That Affects Properties

```java
package sesbeans.circle;

import java.beans.*;

public class CircleBeanBeanInfo extends SimpleBeanInfo {
    
    public PropertyDescriptor[] getPropertyDescriptors() {
        try {
            PropertyDescriptor props[] = {
                new PropertyDescriptor("radius", CircleBean.class)
            };

            props[0].setDisplayName("Radius of circle");
            props[0].setBound(true);

            return props;
        }
        catch (IntrospectionException ex) {
            ex.printStackTrace();
            return super.getPropertyDescriptors();
        }
    }
}
```
A BeanInfo Class That Affects Properties

- Overriding the `getPropertyDescriptors` method
- Limiting visible properties
- `PropertyDescriptor` class
- `setDisplayName` method
- `setBound` method
Using

getAdditionalBeanInfo

• How to use it

```java
// Replace CircleBeanBeanInfo.java with the contents of this file
package sesbeans.circle;

import java.beans.*;

public class CircleBeanBeanInfo extends SimpleBeanInfo {
    public BeanInfo[] getAdditionalBeanInfo() {
        return new BeanInfo[] {
            new CircleBeanAdditionalInfo()
        };
    }
}

class CircleBeanAdditionalInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        try {
            PropertyDescriptor props[] = {
                new PropertyDescriptor("radius", CircleBean.class)
            };
            props[0].setDisplayName("Radius of circle");
            props[0].setBound(true);
            return props;
        } catch (IntrospectionException ex) {
            ex.printStackTrace();
            return super.getPropertyDescriptors();
        }
    }
}
```
BeanInfo Class That Affects Methods

- Limit number of methods for target bean

```java
package sesbeans.stock;

import java.beans.*;
import java.lang.reflect.Method;

public class StockDetailBeanInfo extends SimpleBeanInfo {
    MethodDescriptor method(String name, Class arg) throws NoSuchMethodException {
        Method m = StockDetail.class.getMethod(name, new Class[] {arg});
        return new MethodDescriptor(m);
    }

    public MethodDescriptor[] getMethodDescriptors() {
        try {
            MethodDescriptor[] {method("priceChange", StockPriceChangeEvent.class)};
        } catch(NoSuchMethodException ex) {
            ex.printStackTrace();
            return super.getMethodDescriptors();
        }
    }
}
```
How Is a BeanInfo Processed?

- Probe BeanInfo classes for descriptor information
- Apply reflection and naming conventions
Available BeanInfo Methods

In addition to the methods for the BeanInfo interface, refer to the tables of methods for

- BeanDescriptor class
- EventSetDescriptor class
- PropertyDescriptor class
- MethodDescriptor class
- FeatureDescriptor class
Reflection and JavaBeans

- Advantages of reflection
- Major classes of the Reflection API
Check Your Progress

• List several ways in which supplying a class for a bean can improve its usability

• Describe how the process of introspection works

• Create BeanInfo classes for bean components
Think Beyond

This module has given you a foundation. The following modules build on many of the methods and classes that were discussed in the course of learning about introspection:

• Module 7 – Persistence, how beans are reloaded from a serialized state

• Module 8 and 9 – How to write a customizer and a property editor

• Module 12 – How to build programs that use beans outside of the BeanBox
Module 7

Persistence
Module Overview

• Objectives
• Relevance
• References
Goals for Bean Storage

• General Java object storage
  • Object Serialization API
  • Definition of object serialization

• Bean storage
  • Object serialization mechanism
  • Externalization mechanism
Java Object Serialization

• Serializable interface

• Classes that are serialized
What Is and Is Not Saved

- Bean properties and internal state
- Pointers and event adapters
- Fields automatically serialized
- Fields not serialized
- The keyword `transient`
Input and Output Interfaces

- `ObjectOutput` and `ObjectInput` interfaces
Saving Beans to Streams

- **ObjectOutputStream class**
- **writeObject() method**
- **Example of code syntax**

```java
public static void main(String args[]) {
    String serFile = "day.ser";
    try {
        Date today = new Date();
        FileOutputStream fos = new FileOutputStream(serFile);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(today);
        oos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```
Retrieving Beans From Streams

• `ObjectInputStream` class provides for deserializing persisted objects

• `readObject()` method deserializes objects

```java
public static void main(String args[]) {
    try {
        FileInputStream fis = new FileInputStream("day.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);

        Date yesterday = (Date) ois.readObject();
        System.out.println("The date was: "+ yesterday);

        ois.close();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException ex) {
        // thrown if cannot locate a class used by serialized object
        ex.printStackTrace();
    }
}
```
defaultReadObject and defaultWriteObject Methods

• Private methods

```java
private void writeObject(ObjectOutputStream ostr) throws IOException {
    code
}
```
```java
private void readObject(ObjectInputStream instr) throws IOException {
    code
}
```

• Invoking defaultReadObject and defaultWriteObject
**defaultReadObject and defaultWriteObject Methods**

```java
private void writeObject(ObjectOutputStream ostr) throws IOException {
    // specific definitions

    // call defaultWriteObject method to perform default serialization
    ostr.defaultWriteObject();

    // write your specific information
}

private void readObject(ObjectInputStream instr) throws IOException {
    // specific definitions

    // call defaultReadObject method to perform default deserialization
    instr.defaultReadObject();

    // read back your specific information
}
```
```java
import java.io.*;
import java.awt.Color;

public class Car implements Serializable {

    private Color bodyColor;

    public Car() {
        this(Color.black);
    }

    public Car (Color c) {
        bodyColor = c;
    }

    // simple color property

    public Color getColor() {
        return bodyColor;
    }

    public void setColor (Color c) {
        bodyColor = c;
    }
}
```
```java
// for each time the car is written, also serialize its green color
// component

private void writeObject(ObjectOutputStream oos) {
    try {
        // use the default writing mechanism
        oos.defaultWriteObject();

        // write anything else you need to serialize
        oos.writeInt(bodyColor.getGreen());

        System.out.println("car written: "+ bodyColor);
    } catch(Exception e) {} } } } } } } } 

// for each deserialization, also read the int that was written

private void readObject(ObjectInputStream ois) {
    int i=0;

    try {
        // restore the serialized data values for this object
        ois.defaultReadObject();

        // also read other information that was serialized manually
        i = ois.readInt();

        System.out.println("car read: " + bodyColor);
    } catch(Exception e) {} } } } } } } 
```

---

Sun Educational Services

Sample Code

JavaBeans Component Development

Module 7, slide 12 of 22

Copyright 1999 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services November 1999 Revision B.2
```java
import java.awt.Color;
import java.io.*;
import java.beans.*;

public class SerlCar {
    private Car accord;
    private FileOutputStream fos;
    private ObjectOutputStream oos;
    private FileInputStream fis;
    private ObjectInputStream ois;

    public SerlCar() {
        accord = new Car();
        accord.setColor(Color.green);
    }

    public void writeCar() {
        try {
            fos = new FileOutputStream("Car.ser");
            oos = new ObjectOutputStream(fos);
            oos.writeObject(accord);
            oos.close();
        } catch (IOException e) { }
    }
}
```
```java
31 public void readCar() {
32    try {
33      fis = new FileInputStream("Car.ser");
34      ois = new ObjectInputStream(fis);
35      accord = (Car)ois.readObject();
36      ois.close();
37    } catch (Exception excep) {
38      System.out.println(excep);
39    }
40  }
41
42  public void getBean() {
43    ClassLoader cl = SerlCar.class.getClassLoader();
44    Car myCar = null;
45    try {
46      myCar = (Car)Beans.instantiate(cl, "Car");
47    } catch (Exception excep) {
48      System.out.println(excep);
49    }
50    System.out.println("deserialized bean: " + myCar.getColor());
51  }
52
53  public static void main(String[] args) {
54    SerlCar sc = new SerlCar();
55    sc.writeCar();
56    sc.readCar();
57    sc.getBean();
58  }
59```
Sample Code Discussion

• `Beans.instantiate` method
  • Arguments
  • Steps performed

• Methods invoked by `main`
Deserialization and `Beans.instantiate`

- Bean is serialized to a `.ser` file
- `Beans.instantiate` method creates an instance of the bean
- Bean instance is created by deserializing the `.ser` file
Creating a Java Beans Prototype

- You can create prototypes of Java beans using serialization

- You create a prototype by
  - Serializing a bean to a file
  - Providing a manifest file
  - Building a JAR file containing the serialized bean and the manifest file

- You can customize an existing bean and reuse its state without writing *any* code
Packaging a Prototype Bean

1. Create an instance of the JellyBean bean.
2. Change the foreground color.
3. Serialize the bean to a file.
4. Create a manifest file for the prototype.
5. Package the bean into a JAR file.
6. Clear the BeanBox and load the new JAR file.
7. Create an instance of the prototype bean.
Recap of Persistence

Writing to a serialized stream

```java
String serFile = "filename.ser";
try {
    FileOutputStream fileOutputStream = new FileOutputStream(serFile);
    ObjectOutputStream os = new ObjectOutputStream(fileOutputStream);
    os.writeObject(yourBeanObject);
    os.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Reading from a serialized stream

```java
String serFile = "filename.ser";
try {
    FileInputStream fileInputStream = new FileInputStream(serFile);
    ObjectInputStream is = new ObjectInputStream(fileInputStream);
    BeanType yourBeanRestored = (BeanType) is.readObject();
    is.close();
} catch (Exception e) {
    e.printStackTrace();
}
```
Exercise: Creating a New Bean Through Serialization

- Objective
- Tasks
- Exercise summary
Check Your Progress

• Evaluate the goals for bean storage
• Describe how automatic Java serialization works
• Evaluate which data fields need to be marked as transient using serializable rules
• Describe the requirements to create persistent storage of objects
• Add the mechanisms to persist a bean component
Think Beyond

The BeanBox provides a Properties window for modifying the properties of a bean and a set of property editors.

- Can you define your own property editor or property sheet?

- Is there some other way to customize a bean?
Module 8

Property Sheets and Property Editors
Module Overview

- Objectives
- Relevance
- References
What Can You Do Through Customization?

• What is the definition of customization?
• How do previous modules relate to customization?
  • Property sheets
• Can you extend editor support to new property types?
  • Property editors
• When is a property-specific editor not enough?
  • Customizers
Property Sheets and Property Editors

- Property sheets
  - Definition
  - How property sheets work – builder tools

- Property editors
  - Associated with properties for editing
  - Editors provided by the API
Representing Properties

• Using text fields
• Using choices
• Using a custom dialog
Review of Views

- Molecule
  - moleculeName
  - HyaluronicAcid
  - benzene
  - buckminsterfullerine
  - cyclohexane
  - ethane
  - water

- OurButton
  - fontSize: 1.25
  - background
  - debug:
    - false
  - largeFont:
    - false
  - foreground
  - label: press
  - font: A, B, C, D

Dialog window

- SunJavaBeansuntime
  - dialog:
    - true
    - false
  - min
  - max
  - value
Property Editor Basics

• Description of the property editor API
  • PropertyEditor interface
  • PropertyEditorSupport class
Behavior Requirements

- Support one of three display styles
- Support `setValue`
- Support `PropertyChangeListener`
- Fire `PropertyChangeEvent` when property changes
- Define a default constructor
Overview of All Methods

- `add/removePropertyChangeListener`
- `get/setAsText`
- `getCustomEditor`
- `getJavaInitializationString`
- `getTags`
- `get/setValue`
- `isPaintable`
- `paintValue`
- `supportsCustomEditor`
Bean, Builder Tool, Property Editor, and User Interaction
Predefined or Your Own Editor?

• Builder tools provide a set of predefined editors

• Build your own editor if predefined editors do not match the needs for a particular property
  
  • `PropertyEditor` interface
  
  • `PropertyEditorSupport` class
PropertyEditor Requirements

- Use null argument constructor
- Support `setValue`
- Add and remove property change listeners
package sesbeans.labelEditor;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;

public class LabelEditor implements PropertyEditor {
    private String value;

    //Property editors should provide a default constructor.
    public LabelEditor() {
    }

    // Maintain the property value for the property being edited.
    public void setValue(Object valFromBT) {
        value = (String)valFromBT;
    }

    //Return the value of the property.
    public Object getValue() {
        return value;
    }

    ...
Custom GUI

• Methods required
  • isPaintable
  • paintValue
  • supportsCustomEditor
  • getCustomEditor
7 public class LabelEditor implements PropertyEditor {
   ...

23   // Return true if the class provides a non-empty paintValue method.
24   public boolean isPaintable() {
25      return true;
26   }

28   // Paint a representation of the value into the given area of screen
29   // real estate. Note that the property editor is responsible for
30   // doing its own clipping to fit the drawing into the given region.
31   public void paintValue(Graphics gfx, Rectangle box) {
32      gfx.setClip(box);
33      FontMetrics fm = gfx.getFontMetrics();
34      gfx.drawString("Click to edit...",
35          box.x + 5,
36          (box.y + box.height + fm.getAscent())/2);
37   }

45   // Return the property value as a string, converting from the
46   // property’s data type when necessary.
47   public String getAsText() {
48      return value;
49   }

51   // Set the property value by parsing the given string, converting
52   // it to the property’s data type when necessary.
53   public void setAsText(String text) {
54      value = text;
55   }

57   // If the property value is one of a set of known tagged values,
58   // this method returns an array of the tags. When necessary, the
59   // tags are generated by converting the property data type’s values
60   // to strings.
61   public String[] getTags() {
62      return null;
63   }

//Return a custom GUI component that is used to edit the property value.

public Component getCustomEditor() {
    final TextArea labelArea = new TextArea(value);
    labelArea.setSize(300, 150);
    labelArea.addTextListener(new TextListener()
        public void textValueChanged(TextEvent e) {
            value = labelArea.getText();
            // Notify listeners that the value for this property has been modified.
            listeners.firePropertyChange(null, null, null);
        }
    );
    return labelArea;
}

//Return true if the class provides a custom editor.
public boolean supportsCustomEditor() {
    return true;
}
Choice of Tags

- `getTags` returns a non-null `String[]` of choices
- `BoolEditor` is an example of this type of editor
package sun.beans.editors;

// Property editor for a java built-in "boolean" type.
import java.beans.*;
public class BoolEditor extends PropertyEditorSupport {
    public String getJavaInitializationString() {
        // This must return local independent Java.
        if (((Boolean)getValue()).booleanValue()) {
            return ("true");
        } else {
            return ("false");
        }
    }
    public String getAsText() {
        if (((Boolean)getValue()).booleanValue()) {
            return ("True");
        } else {
            return ("False");
        }
    }
    public void setAsText(String text) {
        if (text.toLowerCase().equals("true")) {
            setValue(Boolean.TRUE);
        } else if (text.toLowerCase().equals("false")) {
            setValue(Boolean.FALSE);
        } else {
            throw new java.lang.IllegalArgumentException(text);
        }
    }
    public String[] getTags() {
        String result[] = { "True", "False" };  
        return result;
    }
}
Simple String in a Text Field

• The least complex display style
• Return non-null String from getAsText
• Support setAsText
StringEditor From the BeanBox

```java
package sun.beans.editors;

import java.beans.*;

public class StringEditor extends PropertyEditorSupport {

    public String getJavaInitializationString() {
        return "\" + getValue() + "\";
    }

    public void setAsText(String text) {
        setValue(text);
    }
}
```
Making Your Property Editor Known

- PropertyEditorManager class
- Naming conventions
- BeanInfo file
import java.beans.*;
import java.awt.*;

public class MultilineLabelBeanInfo extends SimpleBeanInfo {  
  PropertyDescriptor property(String name, String desc)  
      throws IntrospectionException {  
    PropertyDescriptor p = new PropertyDescriptor(name,  
        MultilineLabel.class);  
    p.setShortDescription(desc);  
    return p;  
  }  

  public PropertyDescriptor[] getPropertyDescriptors() {  
    try {  
        PropertyDescriptor props[] = {  
            property("label", "The contents of the control"),  
            property("eolStyle",  
                "The method used to determine end of lines"),  
        };  
        props[0].setPropertyEditorClass(LabelEditor.class);  
        props[1].setPropertyEditorClass(EolStyleEditor.class);  
        props[2].setPropertyEditorClass(AlignmentStyleEditor.class);  
        return props;  
    } catch(IntrospectionException ex) {  
        ex.printStackTrace();  
        return super.getPropertyDescriptors();  
    }  
  }
}
## Recap of Property Editors

### Creating your own property editor

Decide what display style your editor will use.

Define the appropriate methods for that display style: `isPaintable`, `supportsCustomEditor`, `getAsText`, `setAsText`, or `getTags`.

Define a `BeanInfo` file to register your property editor, if necessary.

### Implement `PropertyEditor` interface

Provide a null argument constructor.

Define the `setValue` (Object o) method.

Support the addition and removal of `PropertyChangeListeners`:

```java
private PropertyChangeSupport support = new PropertyChangeSupport(this);

public void addPropertyChangeListener(PropertyChangeListener pl){
    support.addPropertyChangeListener(pl);
}

public void removePropertyChangeListener
    (PropertyChangeListener pl) {
    support.removePropertyChangeListener(pl);
}
```

Define the methods specified by the interface (some may be no-ops).
Exercise: Creating a Property Editor

- Objective
- Preparation
- Tasks
- Exercise summary
Check Your Progress

- Identify the mechanisms provided in the JavaBeans API that enable properties of a bean to be manipulated
- Compare property sheets and property editors
- Create a property editor for a specified property of a bean
Think Beyond

You have a complex bean and have decided that a property editor does not provide the level of help you feel is necessary for anyone customizing your bean.

How do you create a wizard-type aid for users to handle your bean customization?
Module 9

Customizers
Module Overview

• Objectives
• Relevance
• References
When Is a Property-Specific Editor Not Enough?

- Are property editors sufficient to support complex, industrial-strength beans?

- What if a single root choice about the type of the bean rendered half the properties irrelevant?

The JavaBeans specification provides customizers for these wizard-like needs.
Customizers

• Customizer use
• Characteristics of customizers
• Naming conventions
Implementing a Customizer Class

- Defining BeanInfo
- Extending Component
- Implementing Customizer
- Providing a null argument constructor
- Adding and removing PropertyChangeListener
- Defining setObject()
Defining `BeanInfo`

```java
import java.beans.*;

public class MyBeanBeanInfo extends SimpleBeanInfo() {
    public BeanDescriptor getBeanDescriptor() {
        return new BeanDescriptor(MyBean.class, MyBeanCustomizer.class);
    }
}
```
Extending Component/Implementing Customizer

- Extend Component or its subclasses
- Implement the Customizer interface
- Use a null argument constructor

```java
import java.beans.*;

public class MyBeanCustomizer extends Panel implements Customizer {
    public MyBeanCustomizer() {
        // Code to build and add GUI of the customizer
    }
    // All the other methods and code...
}
```
// In the most common case the support can be set globally this way.
private PropertyChangeSupport support = new PropertyChangeSupport(this);

public void addPropertyChangeListener(PropertyChangeListener pl) {
    support.addPropertyChangeListener(pl);
}

public void removePropertyChangeListener(PropertyChangeListener pl) {
    support.removePropertyChangeListener(pl);
}
Defining setObject ()

private MyBean bean;
public void setObject(Object beanToCustomize) {
    bean = (MyBean) beanToCustomize; // cast object passed to correct type
    // Code to get current properties for "bean" and/or
    // Code to build GUI elements for customizer and add them to layout.
}
Example of a Customizer

```java
package sunw.demo.buttons;

import java.awt.*;
import java.awt.event.*;
import java.beans.*;

public class ExplicitButtonCustomizer extends Panel implements Customizer, KeyListener {
    public ExplicitButtonCustomizer() {
        setLayout(null);
    }

    public void setObject(Object obj) {
        target = (ExplicitButton) obj;
        Label t1 = new Label("Caption:", Label.RIGHT);
        add(t1);
        t1.setBounds(10, 5, 60, 30);

        labelField = new TextField(target.getLabel(), 20);
        add(labelField);
        labelField.setBounds(80, 5, 100, 30);

        labelField.addKeyListener(this);
    }

    public Dimension getPreferredSize() {
        return new Dimension(200, 40);
    }

    public Dimension preferredSize() {
        return getPreferredSize();
    }
}
```
Example of a Customizer

```java
40    public void keyTyped(KeyEvent e) {
41    }
42
43    public void keyPressed(KeyEvent e) {
44    }
45
46    public void keyReleased(KeyEvent e) {
47        String txt = labelField.getText();
48        target.setLabel(txt);
49        support.firePropertyChange("", null, null);
50    }
51
52    //------------------------------------------------------------------
53
54    public void addPropertyChangeListener(PropertyChangeListener l) {
55        support.addPropertyChangeListener(l);
56    }
57
58    public void removePropertyChangeListener(PropertyChangeListener l) {
59        support.removePropertyChangeListener(l);
60    }
61
62    private PropertyChangeSupport support = new
63    PropertyChangeSupport(this);
64
65    //------------------------------------------------------------------
66
67    private ExplicitButton target;
68    private TextField labelField;
69    }
```
Recap of Customizers

Creating a customizer class

Define a BeanInfo file

```java
public class MyBeanBeanInfo extends SimpleBeanInfo() {
    public BeanDescriptor getBeanDescriptor() {
        return new BeanDescriptor(MyBean.class, MyBeanCustomizer.class);
    }
}
```

Extend `java.awt.Component` or a subclass and implement the Customizer interface

```java
public class MyBeanCustomizer extends Panel implements Customizer {

    // code here
}
```

Provide a null argument constructor for the customizer

```java
public MyBeanCustomizer() {
    // code here
}
```

Support the addition and removal of a PropertyChangeListener

```java
private PropertyChangeListener support = new PropertyChangeListener(this);

public void addPropertyChangeListener(PropertyChangeListener pl) {
    support.addPropertyChangeListener(pl);
}

public void removePropertyChangeListener(PropertyChangeListener pl) {
    support.removePropertyChangeListener(pl);
}
```

Define the setObject(Object bean) method

```java
private MyBean bean;
public void setObject(Object bearToCustomize) {
    bean = (MyBean) bearToCustomize;
    // remaining code here
}
```
Exercise: Creating a Customizer

• Objective
• Preparation
• Tasks
• Exercise summary
Check Your Progress

• Describe a situation where it would be beneficial to use a customizer for a bean

• Create a customizer for a bean component
Think Beyond

The next module on adapters addresses some of the issues that can occur when trying to handle events for which an object has registered.

- For example, if you register with several components that send ActionEvents, how do you determine which component the ActionEvent originated from?

- What about the case where an event source has several interested objects that want customized information from the event sent by the source?
Module 10

Event Adapters
Module Overview

- Objectives
- Relevance
- References
What Is an Event Adapter?

- Definition
- Diagram overview
Adapters Used in the BeanBox

• When are hookup classes created?
• Where is the generated code stored?
• Example of an adapter hookup class:

```java
package tmp.sun.beanbox;

public class ___Hookup_13fdc34850 implements java.awt.event.ActionListener,
java.io.Serializable {
    public void setTarget(sl291.choice.MyChoice t) {
        target = t;
    }

    public void actionPerformed(java.awt.event.ActionEvent arg0) {
        target.addRemoveItem(arg0);
    }

    private sl291.choice.MyChoice target;
}
```
Types of Adapters

- Demultiplexing
- Multiplexing
- Common uses of adapters
Adapter Diagrams

Diagram 1:
- Event source 1
  - Fire event
- Event source 2
  - Fire event
- Event source 3
  - Fire event

Diagram 2:
- Event source
  - Fire event
- Multiplexing adapter
  - Forward event or call specific method
  - Interested object

Diagram 3:
- Multiplexing adapter
  - Fire event
  - Interested object
  - Interested object
  - Interested object
Differentiating Adapters From Normal Listeners

• Case with no adapters
• Case with adapters
Demultiplexing Adapter Example

• Application example

• Description of application
  • Builder.java
  • Widgets.java
  • ActionAdapter.java
  • WhatToDo.java
Description of Application

![Diagram showing relationship between Source and Target with widgets and actions]

- Source:
  - Widgets
    - Apply
    - Quit
    - Fonts
    - Color
    - Patterns

- Target:
  - WhatToDo
    - doApply
    - doQuit
    - doFonts
    - doColor
    - doPatterns

Builder:
- ActionAdapter(src, targ)

Create instance of ActionAdapter

Create instance of ApplyAdapter, QuitAdapter, ListAdapter
package sesbeans.widadapter;

public class Builder {

    // The builder is responsible for attaching the source to the
    // adapter to the target.

    public static void main(String[] args) {
        Widgets src = new Widgets();
        WhatToDo targ = new WhatToDo();

        // To connect the pieces, make the adapter which knows about
        // both the source and the target
        ActionAdapter adapter = new ActionAdapter(src, targ);
    }
}
Widgets.java

```java
package sesbeans.widadapter;

import java.awt.*;
import java.awt.event.*;

// Create the Widgets for this application
// Give any listener the opportunity to choose
// any or all of the widgets to listen to.

public class Widgets {

    private Frame fr;
    private Button apply;
    private Button quit;
    private List myList;

    public Widgets() {
        // Make the GUI with its widgets
        fr.add(apply);
        fr.add(quit);
        fr.add(myList);

        // Put up the frame
        fr.pack();
        fr.setVisible(true);
    }

    public void addApplyListener(ActionListener targ) {
        apply.addActionListener(targ);
    }

    public void removeApplyListener(ActionListener targ) {
        apply.removeActionListener(targ);
    }
}
```
Widgets.java

```java
49
50 public void addQuitListener(ActionListener targ) {
51    quit.addActionListener(targ);
52 }
53
54 public void removeQuitListener(ActionListener targ) {
55    quit.removeActionListener(targ);
56 }
57
58 public void addListListener(ActionListener targ) {
59    myList.addActionListener(targ);
60 }
61
62 public void removeListListener(ActionListener targ) {
63    myList.removeActionListener(targ);
64 }
65
66 // Typical WindowAdapter to help kill the application
67 class WL extends WindowAdapter {
68    public void windowClosing(WindowEvent e) {
69        // kill the AWT thread by killing the application
70        System.exit(0);
71    }
72 }
73 }
```
package sesbeans.widadapter;

import java.awt.*;
import java.awt.event.*;

// An adapter that receives ActionEvents from widgets in
// the source bean. Not only does the adapter call
// appropriate methods in the target, it can do any other
// processing needed; in this case, it simply shows or
// hides a frame when one of the source buttons is pushed.

public class ActionAdapter {
    private Widgets src;
    private WhatToDo targ;

    public ActionAdapter(Widgets s, WhatToDo t) {
        src = s;
        targ = t;

        // We're choosing to make a different subtype of adapter
        // for each source. That didn't have to be the case, but
        // it does make for cleaner actionPerformed() methods.
        src.addApplyListener(new ApplyAdapter());
        src.addQuitListener(new QuitAdapter());
        src.addListListener(new ListAdapter());
    }

    class ApplyAdapter implements ActionListener {
        public void actionPerformed(ActionEvent evt) {
            // Since it came from the Apply in the source,
            // do something interesting like displaying an icon
            displayIcon();
            // Call target's appropriate method; could have passed
            // information, had it been needed
            targ.apply();
        }
    }
}
ActionAdapter.java

Code

39
40 class QuitAdapter implements ActionListener {
41    public void actionPerformed(ActionEvent evt) {
42        // Get rid of the icon since you're quitting
43        fr.setVisible(false);
44        // Call target's appropriate method
45        targ.quit();
46    }
47 }
48
49 class ListAdapter implements ActionListener {
50    public void actionPerformed(ActionEvent evt) {
51        // No need to do anything more than call the right
52        // target method.
53        List l = (List)evt.getSource();
54        int index = l.getSelectedIndex();
55
56        if (index == 0) {
57            targ.doFonts();
58        } else if (index == 1) {
59            targ.doColor();
60        } else if (index == 2) {
61            targ.doPatterns();
62        } else {
63            System.out.println("Error: item not recognized");
64        }
65    }
66 }
67
68
69    // The displayIcon() method

98    //
package sesbeans.widadapter;

import java.awt.*;

// The target has several jobs that it can do

public class WhatToDo {

    public void apply() {
        System.out.println("Do apply ...");
    }

    public void quit() {
        System.out.println("Quitting...");
    }

    public void doFonts() {
        System.out.println("Put up Fonts dialog");
    }

    public void doColor() {
        System.out.println("Put up Color dialog");
    }

    public void doPatterns() {
        System.out.println("Put up Patterns dialog");
    }
}
Multiplexing Adapters

Overview of multiplexer exercise

- Receives DataEvent from test button
- Generates DataTableEvent
- Fires DataTableEvent to
  - XAxis DataTableBean
  - YAxis DataTableBean
  - Table DataTableBean
Overview of Multiplexer

Exercise
Exercise: Working With Adapters

• Objective
• Preparation
• Tasks
• Exercise summary
Check Your Progress

- Define event adapter
- Compare multiplexing and demultiplexing adapters
- List the common uses of event adapters
- Differentiate an event adapter from an event listener
- Write an event adapter for a bean component
Think Beyond

- How can beans be used in distributed systems?
- Can beans be used as intelligent front-end clients that interface with network servers?
- How might this work with RMI, JDBC™, and JavaIDL APIs from JavaSoft™?
Module 11

Distributed Computing
With Beans
Module Overview

• Objectives
• Relevance
• References
Distributed Bean Programming

• Areas of familiarity
  • Facilities of the Java programming language
  • RMI API
  • Architecture options for distributed object solutions
  • Technologies for enterprise services and transactions
Enterprise JavaBeans

• Synopsis

• Enterprise JavaBeans specification
  • Component architecture for distributed, object-oriented, business applications
  • Framework for the deployment of enterprise components
EJB Developer Roles

- Server/container developer
- EJB developer
- Deployer
- Application assembler
- System administrator
EJB Features

- Multi-tier, distributed architecture
- EJB server support
- Flexible and extensible components
- Protocol support
Types of Enterprise Beans

- Session
- Entity
JavaBeans and Enterprise JavaBeans
## JavaBeans and Enterprise JavaBeans

<table>
<thead>
<tr>
<th>JavaBeans</th>
<th>Enterprise JavaBeans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual and deployed on client-side, usually</td>
<td>Non-visual and deployed on server-side, always</td>
</tr>
<tr>
<td>Deployed like any class – as an applet or application</td>
<td>Deployed in a container</td>
</tr>
<tr>
<td>Properties and behaviors usually introspected by builder tool</td>
<td>Properties and context discovered from deployment descriptor file</td>
</tr>
<tr>
<td>An unmanaged component</td>
<td>A component managed by the container and server</td>
</tr>
<tr>
<td>Events used extensively</td>
<td>Events not used, normally</td>
</tr>
</tbody>
</table>
Applications for Distributed Beans

• Workflow applications
• Application servers
• Agents
Distributed Computing Technologies

- JDBC
- RMI
- JavaIDL
• **QuoteMonitorBean**
Definition of RMI

• A set of classes and interfaces designed to enable you to make calls to remote objects that exist in the runtime of a different Java virtual machine (JVM) invocation

• A Java-to-Java mechanism
RMI Architecture Overview

- How it works
- Graphical overview
RMI Architecture Overview

• Transport layer
• Remote Reference layer
• RMI stubs and skeletons
• rmic command
• rmiregistry application
RMI Exercise Code

- Exercise description
- Conceptual overview
Source Files Provided

- **Interfaces:** `DataFactory.java`, `Data.java`
- **Implementations and server:** `DataFactoryImpl.java`, `DataImpl.java`, `DataServer.java`
- **Conceptual overview of exercise source files:**

![Diagram showing the relationships between DataClient.java, DataFactory.java, Data.java, List of DataImpl IDs, DataServer.java, and rmiregistry & java DataServer.]
package sesbeans.rmilab;

import java.rmi.*;
import java.awt.Dimension;

public interface Data extends Remote{
    public String getID() throws RemoteException;
    public String getXLabel() throws RemoteException;
    public String getYLabel() throws RemoteException;
    public String[] getXAxis() throws RemoteException;
    public String[] getYAxis() throws RemoteException;
    public Dimension getSize() throws RemoteException;
    public String[][] getDataValues() throws RemoteException;
}
package sesbeans.rmilab;

import java.rmi.*;

public interface DataFactory extends Remote{

    public Data getData(String id) throws RemoteException;
    public String[] getDataList() throws RemoteException;
    public void addData(DataImpl dImpl) throws RemoteException;
}
package sesbeans.rmilab;

import java.rmi.*;
import java.rmi.server.*;
import java.awt.Dimension;

public class DataImpl extends UnicastRemoteObject implements Data{

    private String id, xLabel, yLabel;
    private String [] xAxis, yAxis;
    private String [][] dataValues;
    private Dimension size;

    public DataImpl(String id, String xLabel, String yLabel)
    throws RemoteException {
        this.id = id;
        this.xLabel = xLabel;
        this.yLabel = yLabel;
        this.size = new Dimension (8,5);
        xAxis = new String [size.height];
        yAxis = new String [size.width];
        dataValues = new String [yAxis.length] [xAxis.length];
    }

    public String getID() throws RemoteException {
        return id;
    }
}
public String getXLabel() throws RemoteException {
    return xLabel;
}

public String getYLabel() throws RemoteException {
    return yLabel;
}

public String[] getXAxis() throws RemoteException {
    return xAxis;
}

public String[] getYAxis() throws RemoteException {
    return yAxis;
}

public Dimension getSize() throws RemoteException {
    return size;
}

public String[][] getDataValues() throws RemoteException {
    return dataValues;
}

public void setXAxis(String[] sa1) {
    xAxis = sa1;
}

public void setYAxis(String[] sa2) {
    yAxis = sa2;
}

public void setSize(int width, int height) {
    size = new Dimension(width, height);
}

public void setDataValues(String[][] sa3) {
    dataValues = sa3;
}
```java
package sesbeans.rmilab;

import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;

public class DataFactoryImpl extends UnicastRemoteObject implements DataFactory{

    private Vector storage;

    public DataFactoryImpl() throws RemoteException {
        storage = new Vector(1,1);
    }

    public Data getData(String id) throws RemoteException {
        for (int i = 0; i < storage.size(); i++) {
            if (((DataImpl)storage.elementAt(i)).getID().equals(id)){
                return (Data)storage.elementAt(i);
            }
        }
        return null;
    }

    public String [] getDataList() throws RemoteException {
        String [] names = new String [storage.size()];
        for (int i = 0; i < storage.size(); i++) {
            names[i] = ((DataImpl)(storage.elementAt(i))).getID();
        }
        return names;
    }

    public void addData(DataImpl dImpl) throws RemoteException {
        storage.addElement(dImpl);
    }

}
```
Server Code Overview

• Description of the DataServer application

• Description of binding a `DataFactoryImpl` object to a name in `rmiregistry`

```java
Naming.rebind("dataServer", dfi);
```
Exercise: Creating an RMI Client Bean

• Objective
• Preparation
• Tasks
• Exercise summary
Check Your Progress

- Describe the key features of Enterprise JavaBeans
- Describe, in one paragraph, the Java RMI architecture
- Explain how to implement a Java RMI bean
- Create bean components to be used on the client side of a Java RMI application
Think Beyond

You have been running and testing bean components using the BeanBox. The BeanBox is not considered an application builder tool.

Without such a tool, how do you use your beans and build an application that runs outside of the BeanBox?
Module 12

Beans Outside the BeanBox
Module Overview

• Objectives
• Relevance
• References
Options for Building Beans

- Building a bean from scratch
- Inheriting from a bean to form a new bean
- Subclassing a bean and adding a `BeanInfo` file
- Composing a bean from other beans
- Instantiating a bean from a serialized prototype
Subclassing a Bean and Adding BeanInfo

```java
public class ExplicitButton extends OurButton {
}
```

- Add a BeanInfo to
  - Restrict visible properties
  - Add icons
  - Specify a customizer
  - Limit visible events on the Edit menu
Restricting Visible Properties

```java
12 public PropertyDescriptor[] getPropertyDescriptors() {
13     try {
14         PropertyDescriptor background =
15             new PropertyDescriptor("background", beanClass);
16         PropertyDescriptor foreground =
17             new PropertyDescriptor("foreground", beanClass);
18         PropertyDescriptor font = new PropertyDescriptor("font",
19             beanClass);
20         PropertyDescriptor label =
21             new PropertyDescriptor("label", beanClass);
22
23         background.setBound(true);
24         foreground.setBound(true);
25         font.setBound(true);
26         label.setBound(true);
27
28         PropertyDescriptor rv[] = {background, foreground, font, label};
29         return rv;
30     } catch (IntrospectionException e) {
31         throw new Error(e.toString());
32     }
33 }
```
Specifying a Customizer for the Bean

```java
64 public BeanDescriptor getBeanDescriptor() {
65     BeanDescriptor back = new BeanDescriptor(beanClass, customizerClass);
66     back.setValue("hidden-state", Boolean.TRUE);
67     return back;
68 }
```

```java
84 private final static Class beanClass = ExplicitButton.class;
85 private final static Class customizerClass = ExplicitButtonCustomizer.class;
```
Adding Icons

70 public java.awt.Image getIcon(int iconKind) {
71    if (iconKind == BeanInfo.ICON_MONO_16x16 ||
72        iconKind == BeanInfo.ICON_COLOR_16x16 ) {
73        java.awt.Image img = loadImage("ExplicitButtonIcon16.gif");
74        return img;
75    }
76    if (iconKind == BeanInfo.ICON_MONO_32x32 ||
77        iconKind == BeanInfo.ICON_COLOR_32x32 ) {
78        java.awt.Image img = loadImage("ExplicitButtonIcon32.gif");
79        return img;
80    }
81    return null;
82 }
Limiting Visible Events

```java
public EventSetDescriptor[] getEventSetDescriptors() {
    try {
        EventSetDescriptor push = new EventSetDescriptor(beanClass,
                "actionPerformed",
                java.awt.event.ActionListener.class,
                "actionPerformed");

        EventSetDescriptor changed = new EventSetDescriptor(beanClass,
                "propertyChange",
                java.beans.PropertyChangeListener.class,
                "propertyChange");

        push.setDisplayName("button push");
        changed.setDisplayName("bound property change");

        EventSetDescriptor[] rv = { push, changed};
        return rv;
    } catch (IntrospectionException e) {
        throw new Error(e.toString());
    }
}
```
Composing a Bean From Other Beans

- Use `Beans.instantiate` instead of `new` with beans
- Make composite bean responsible for
  - Proper reading and writing of all beans
  - Restoring connections between beans if loaded from a persisted state
Customizing and Saving a Bean

```java
import java.beans.*;
import sunw.demo.molecule.*;
import java.io.*;

// Create the serialized file of molecule changed to Benzene
public class MyMolSer {
    private static Molecule moleculeB;

    public MyMolSer() {
        ClassLoader cl = null;
        try {
            cl = MyMolSer.class.getClassLoader();
            moleculeB = (Molecule)
                Beans.instantiate(cl, "sunw.demo.molecule.Molecule");
            moleculeB.setMoleculeName("benzene");
            FileOutputStream fos = new
                FileOutputStream("sunw/demo/molecule/
moleculeSerFile.ser");
            ObjectOutputStream outputStream = new
                ObjectOutputStream(fos);
            outputStream.writeObject(moleculeB);
        } catch (Exception e) {
            throw new Error(e.toString());
        }
    }

    public static void main (String args[]) {
        MyMolSer m = new MyMolSer();
    }
}
```

java MyMolSer
import java.awt.*;
import java.awt.Component;
import java.beans.*;
import java.io.*;

// Read the serialized file and display the bean
public class BeanUnSer extends Frame {
    private static Component myBean;
    public BeanUnSer(String beanSerFile) {
        super("Unserialized from: " + beanSerFile);
        try {
            myBean = (Component) Beans.instantiate(null, beanSerFile);
        } catch (Exception e) {
            System.out.println("Error unserializing bean " + beanSerFile +": " + e);
        }
        this.add(myBean);
        this.setSize(300, 300);
        this.show();
    }

    public static void main (String args[]) {
        BeanUnSer m = new BeanUnSer(args[0]);
    }
}
Creating Applets and Applications With Beans

• Using a builder tool: steps involved
• Coding programmatically
Delivering Your Beans

- A set of `.class` files
- How JAR files are used
- JAR file review
Using JAR Files in HTML

- **APPLET** tag and attributes
  - ARCHIVE
  - OBJECT
  - CODE
  - CODEBASE
Sample Bean Applet

<APPLET
  ARCHIVE=juggler.jar
  CODEBASE=./
  CODE=sunw.demo.juggler.Juggler
  WIDTH=200
  HEIGHT=200
>
</APPLET>
Loading and Instantiating a Bean

• Steps involved

• Coding possibilities
  • From a serialized stream
  • In an applet – applet class loader
  • In an application – bean or system class loader
Instantiating a Bean From a Serialized Stream

The `beanClassName` specified must resolve to a serialized object (a `beanClassName.ser` file must exist)
Instantiating a Bean in an Applet/Application

• In an applet

ClassLoader cl = this.getClass().getClassLoader();
myBean = (MyBean) Beans.instantiate(cl, "myorg.mypkg.MyBean");

• In an application

import myorg.mypkg.MyBean;

ClassLoader cl = Class.forName("myorg.mypkg.MyBean").getClassLoader();
myBean = (MyBean) Beans.instantiate(cl, "myorg.mypkg.MyBean");
Sample Code

- Using a JAR file to instantiate a bean in an application

```java
1 import java.net.*;
2 import java.beans.*;
3
4 public class JarReader {
5
6     // URLClassLoader requires an array of URLs
7     private URL[] regURL = new URL[1];
8     private URLClassLoader urlCL;
9     private Class classFromJar;
10
11    public static void main(String[] args) {
12        JarReader jr = new JarReader();
13        jr.readFromJar();
14    }
15
16    public void readFromJar() {
17
18        // Get a URL to your jar file
19        try {
20            // Use the local URL
21            regURL[0] = new URL("jar:file:widadapter.jar!/");
22            // Could read the file from a different host
23            //regURL[0] = new URL("jar:http://ghost:8080/widadapter.jar!/");
24            } catch (MalformedURLException e) {
25                System.out.println("bad URL: " + e);
26            }
27
28    }
```
```java
27     // From the URL, obtain a class loader
28     urlCL = new URLClassLoader(regURL);
29
30     // Get a class from the jar
31     try {
32         classFromJar = urlCL.loadClass("sesbeans.widadapter.Builder");
33         try {
34             // Prove that everything worked by forcing the loaded
35             // class to be used
36             // Could also do
37             // Object o =(Object)Beans.instantiate(urlCL,
38            // "sesbeans.widadapter.Builder");
39             Object o = classFromJar.newInstance();
40             System.out.println(o);
41         } catch (Exception e) {
42             System.out.println("can't make new instance: " + e);
43         }
44     } catch (ClassNotFoundException e) {
45         System.out.println("bad class load: " + e);
46     }
47
48     // Get a data file from the jar
49     URL gifURL = classFromJar.getResource("dukeJB.gif");
50 }
51 }
52 }
53 ```
Sample Code

• Instantiating a bean in an applet

```java
import sesbeans.twoSimpleBeans;

import java.awt.Frame;
import java.awt.BorderLayout;
import java.applet.Applet;
import java.beans.
import java.io.*;

public class LabelChanger extends Applet {

    public void init() {
        MyButton myButton = null;
        MyText myText = null;
        try {
            ClassLoader cl = this.getClass().getClassLoader();
            myText = (MyText) Beans.instantiate(cl, "sesbeans.twoSimpleBeans.MyText");
            myButton = (MyButton) Beans.instantiate(cl, "sesbeans.twoSimpleBeans.MyButton");
        } catch (Exception e) {
            throw new RuntimeException(e.toString());
        }
    }
}
```
```java
23     myText.addLabelListener(myButton);
24     setLayout(new BorderLayout());
25     add(myText, BorderLayout.CENTER);
26     add(myButton, BorderLayout.SOUTH);
27 }

28 }
29 public static void main(String args[]) {
30     Frame f = new Frame("Label Changer Application");
31     LabelChanger labelChanger = new LabelChanger();
32     f.setLayout(new BorderLayout());
33     f.add(labelChanger, BorderLayout.CENTER);
34     labelChanger.init();
35     f.setSize(300,100);
36     f.setVisible(true);
37 }
38 }
39 } // class LabelChanger
```
The `-jar` Option to the `java` Command

- Run a class from a JAR file that contains a main method using

```java
java -jar jarnameme.jar
```

- Include a manifest file in the JAR containing the `Main-Class` line using

```xml
Main-Class: LabelChanger
```
Hooking Beans Together

- Instantiate the beans using `Beans.instantiate`
- Register listener beans with source beans
  ```java
  sourceBean.addXXXListener(listenerBean)
  ```
- Register visual beans with layout manager
Issues for Applets That Are Beans

• Running an applet bean in a browser: the browser initializes and starts the applet bean

• Instantiating a bean that is an applet with Beans.instantiate in an application

• Creating a bean that is an applet
Instantiating a Bean That Is an Applet

```java
import java.beans.*;
import java.awt.*;
import sunw.demo.juggler.*;

public class TestApplet {
    private Frame fr;
    private Juggler jugg;

    public static void main (String args[]){
        TestApplet tst = new TestApplet();
        tst.doDisplay();
    }

    public void doDisplay() {
        ClassLoader cl;
        try {
            cl = TestApplet.class.getClassLoader();
            jugg = (Juggler)Beans.instantiate(cl, 
                "sunw.demo.juggler.Juggler");
        } catch (Exception e) { 
            throw new Error(e.toString());
        }
        fr = new Frame("test applet instantiation");
        fr.add(jugg);
        jugg.start();
        fr.pack();
        fr.setVisible(true);
    }
}
```
Writing a Bean That Is an Applet

- Do not use an HTML file to pass `PARAM` values
- Provide property `get` and `set` methods instead
- Test using Applet Viewer and BeanBox
Summary of Issues

- Using `init` and `start` for serialized bean applet
- Restoring properties for serialized bean
- Calling `stop` before serializing an applet
- Using `get` and `set` methods for handling `PARAM` values
- Reviewing the life cycle of an applet bean
- Using the `NAME` tag
Exercise: Writing Applets or Applications With Beans

- Objective
- Preparation
- Tasks
- Exercise summary
Check Your Progress

• Provide examples of the various scenarios for building beans

• Create an applet that uses existing bean components and run it in the Applet Viewer

• Create an application that uses bean components
Think Beyond

The next module, describes tools that enable you to use beans in the business environment.

You will also get a preview of the current and future advances that will make beans easier to use in your enterprise solutions.
Module 13

Business Environment for JavaBeans
Module Overview

- Objectives
- Relevance
- References
“Write Once, Run Anywhere”

- Phrase associated with Java language
- With JavaBeans, the phrase has become:

  “Write once, run anywhere, reuse everywhere”
Component-based Software Review

• Business influences

• Major elements
  • Components
  • Containers
  • Scripting

• Review of services
Bridging JavaBeans to Other Component Models

- Other component models in the industry
- Portability to containers
- ActiveX bridge
- JavaBeans Migration Assistant for ActiveX
Development Environments

- Integrated development environments (IDEs)
- Rapid application development (RAD)
Visual Application Builder Tools

- Determining what they are
- Deciding on the tool for you
JavaBeans Added Capabilities

- Glasgow project
- InfoBus technology
- Packaging
Handling or Sharing Data Among Beans

• How BeanContext, JavaBeans Activation Framework (JAF), and InfoBus differ
  • BeanContext – About rendezvous, hierarchy, and services
  • JAF – About associating components with encapsulated MIME typed external data
  • InfoBus – About sharing data between loosely coupled components
InfoBus Technology

- InfoBus technology and JavaBeans
- InfoBus architecture overview
  - Data producers
  - Data consumers
  - Data controllers
Overview of the InfoBus Architecture
InfoBus Classes and Interfaces

• InfoBus class

public final class InfoBus extends java.lang.Object implements java.beans.PropertyChangeListener

static InfoBus get(java.awt.Component component)
static InfoBus get(java.lang.String busName)

• InfoBusMember interface

public abstract interface InfoBusMember

• getInfoBus and setInfoBus methods
• InfoBus property (bound and constrained)
InfoBus Classes and Interfaces

- InfoBusDataProducer interface
  
  public abstract interface InfoBusDataProducer
  extends InfoBusEventListener

- InfoBusDataConsumer interface
  
  public abstract interface InfoBusDataConsumer
  extends InfoBusEventListener
Code Samples From the InfoBus Software

public class RowsetSource extends Applet
    implements InfoBusMember, InfoBusDataProducer

and

public class SimpleConsumerBean extends Applet
    implements InfoBusBean, InfoBusDataConsumer, 
    PropertyChangeListener, 
    DataItemChangeListener,
    java.awt.event.ItemListener

and

public class SampleConsumer extends Applet
    implements InfoBusMember, InfoBusDataProducer, 
    InfoBusDataConsumer, DataItemChangeListener, 
    ActionListener, ItemListener
InfoBus Events

java.util.EventObject

javax.infobus.InfoBusEvent

javax.infobus.ItemAvailableEvent

javax.infobus.ItemRevokedEvent

javax.infobus.ItemRequestedEvent
InfoBus Event Listeners

- InfoBusItemAvailableEvent
- InfoBusItemRevokedEvent
- InfoBusItemRequestedEvent
- Event handlers
Events, Firing Methods, and Handlers

<table>
<thead>
<tr>
<th>Events</th>
<th>InfoBus Methods</th>
<th>Consumer Methods</th>
<th>Producer Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoBusItemAvailableEvent</td>
<td>fireItemAvailable</td>
<td>dataItemAvailable</td>
<td></td>
</tr>
<tr>
<td>InfoBusItemRevokedEvent</td>
<td>fireItemRevoked</td>
<td>dataItemRevoked</td>
<td></td>
</tr>
<tr>
<td>InfoBusRequestedEvent</td>
<td>findDataItem</td>
<td></td>
<td>dataItemRequested</td>
</tr>
<tr>
<td></td>
<td>findMultipleDataItems</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
InfoBus DataItem Interface

- Interface definition
  ```java
  public abstract interface DataItem
  ```

- Access interfaces
  - ImmediateAccess
  - ArrayAccess
  - RowsetAccess
  - ScrollableRowsetAccess
  - DbAccess
Sample Data Items From the InfoBus Software

public class MonetaryDataItem implements ImmediateAccess, DataItem, DataItemChangeManager

public class SimpleDataItem implements ImmediateAccess, DataItem, DataItemChangeManager, InfoBusPropertyMap
Miscellaneous

- DataControllers
- InfoBusPolicyHelper interface
- Guidelines for well-behaved InfoBus components
JavaBeans `beancontext` Package

- Provides a logical, traversable, hierarchy of JavaBeans
- Provides a mechanism for a bean to obtain services from its environment
Beans and BeanContexts
BeanContext Interface

public interface BeanContext extends BeanContextChild, Collection, DesignMode, Visibility {...}

• Methods
  • instantiateChild
  • getResourceAsStream
  • addBeanContextMembershipListener

• Lock – globalHierarchyLock
BeanContextServices Interface

- addService
- revokeService
- hasService
- getService
- releaseService
- getCurrentServiceClasses
- getCurrentServiceSelectors
- add/removeBeanContextServicesListener
Providing a Bean With a BeanContext

Beans.instantiate(classloader cl, String beanName, BeanContext beanCtxt)
BeanContext Support for Applets

`Beans.instantiate(classloader cl, String beanName, BeanContext beanCtxt, AppletInitializer ai)`

- `ai.initialize(applet, beanCtxt)`
- **What conformant implementations of AppletInitializer provide**
BeanContext Services Support

- InfoBus technology
- Printing
- Design/runtime mode
- Bean visibility
- Locale
BeanContext Support Classes

- `java.beans.beancontext.BeanContextChildSupport`
- `java.beans.beancontext.BeanContextSupport`
- `java.beans.beancontext.BeanContextServicesSupport`
JavaBeans Activation Framework

- Determines the type of arbitrary data
- Encapsulates access to data
- Discovers the operations available on data
Major Elements Comprising the JAF Architecture
Overview of the Major Elements

- `DataSource interface`
  - `FileDataSource`
  - `URLDataSource`
- `CommandMap class`
- `CommandObject interface`
Overview of the Major Elements

- **DataHandler class**
  - Retrieves the data typing information
  - Provides a list of commands for data
  - Implements `awt.datatransfer.Transferable`
- **DataContentFactory interface**
- **DataContentHandler interface**
Check Your Progress

- Compare and contrast rapid application development (RAD) and integrated development environment (IDE) software with regard to component development

- Draw a diagram that captures the main interfaces and classes used in the InfoBus

- Draw a diagram that captures the main interfaces and classes used in the beancontext package

- Draw a diagram that captures the main interfaces and classes used in the JavaBeans Activation Framework architecture
Think Beyond

You might now consider taking the Enterprise JavaBeans course from Sun Educational Services. See http://java.sun.com/aboutJava/training/index.html.