JavaBeans Component Development

SL-291

Course Contents

Preface

About This Course

Course Goal

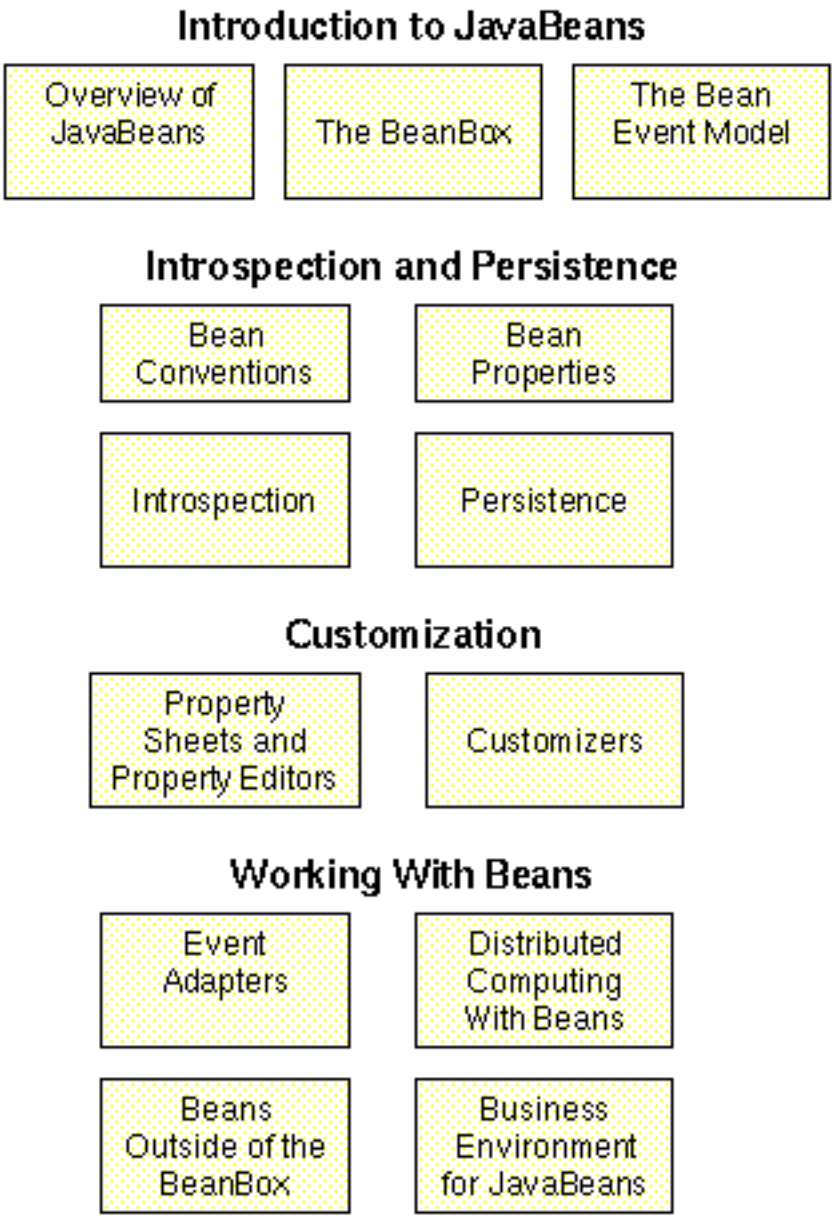
This course provides you with knowledge and skills to

- 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

Course Map



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

Skills Gained	Module			
	1	2	3	4
Skill or Objective 1	■	■		
Skill or Objective 2	■		▒	
Skill or Objective 3		▒	■	■
Skill or Objective 4				

Guidelines for Module Pacing

Module	Day 1	Day 2	Day 3	Day 4
"About This Course"	A.M.			
Module 1 – "Overview of JavaBeans"	A.M.			
Module 2 – "The BeanBox"	A.M/P.M.			
Module 3 – "The Bean Event Model"	P.M.			
Module 4 – "Bean Conventions"		A.M.		
Module 5 – "Bean Properties"		A.M.		
Module 6 – "Introspection"		P.M.		
Module 7 – "Persistence"		P.M.		
Module 8 – "Property Sheets and Property Editors"			A.M.	
Module 9 – "Customizers"			A.M./P.M.	
Module 10 – "Event Adapters"			P.M.	
Module 11 – "Distributed Computing With Beans"				A.M.
Module 12 – "Beans Outside of the BeanBox"				A.M./P.M.
Module 13 – "Business Environment for JavaBeans"				P.M.

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

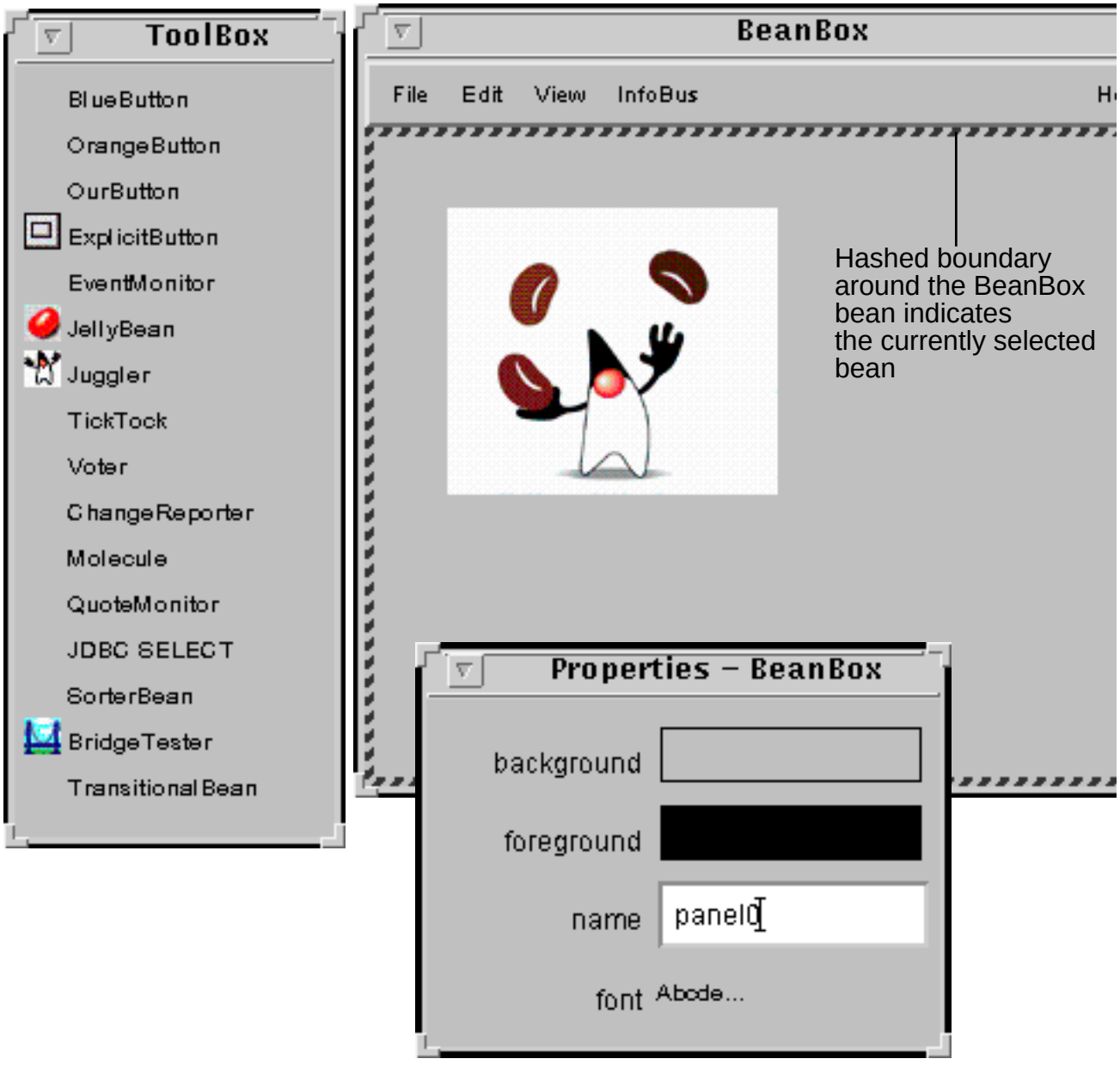
Life Cycle of a Bean

- Development
- Design time
- Run time

The BDK 1.0 Overview

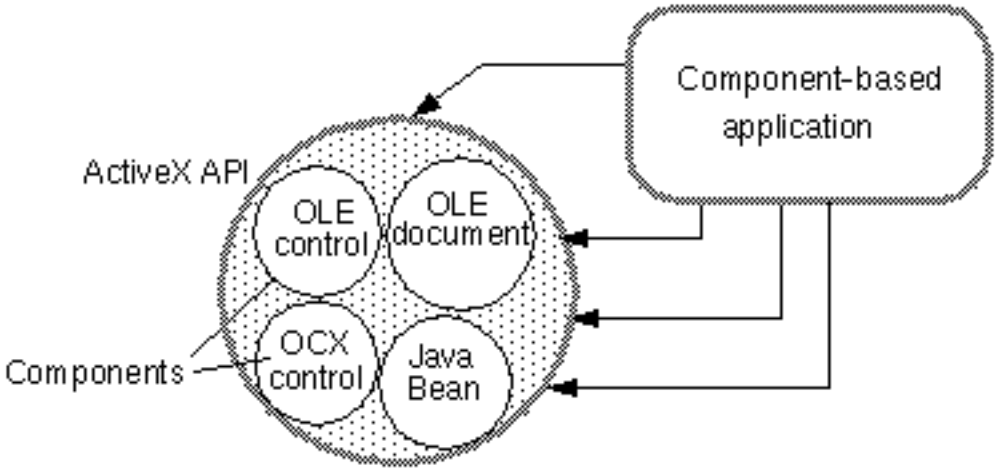
- The BDK provides a reference implementation of the JavaBeans Specification and is intended for bean developers and tool vendors.
- The BDK contains
  - A reference bean container called the BeanBox
  - Sample source code for developing JavaBeans components
  - An on line tutorial for developing JavaBeans
- Use the BDK with JDK™ 1.1.5 or later.

The BDK 1.0 Overview



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

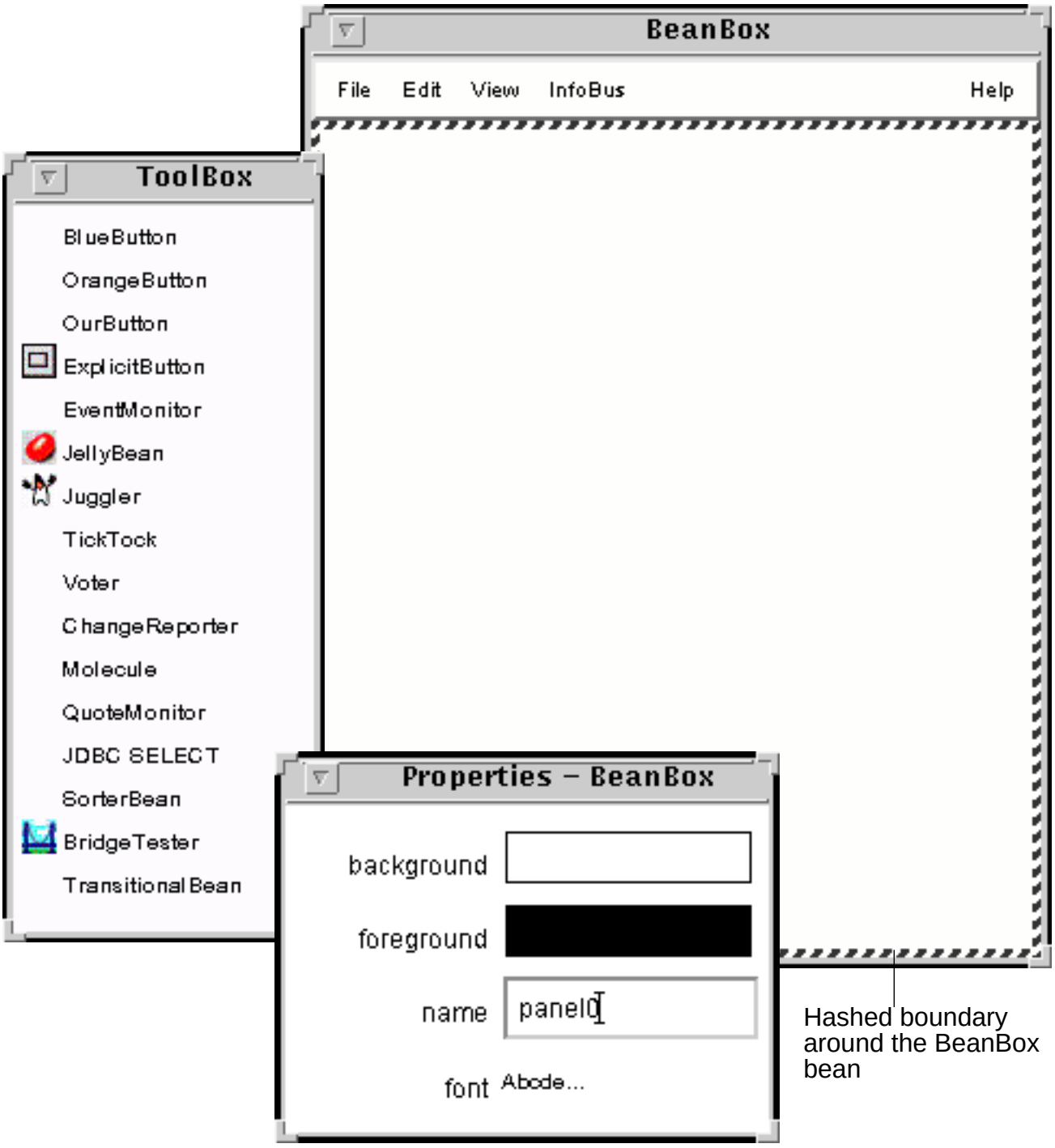
Module Overview

- Objectives
- Relevance
- References

What Is the BeanBox?

- Definition
- Running the BeanBox

Windows of the BeanBox



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.

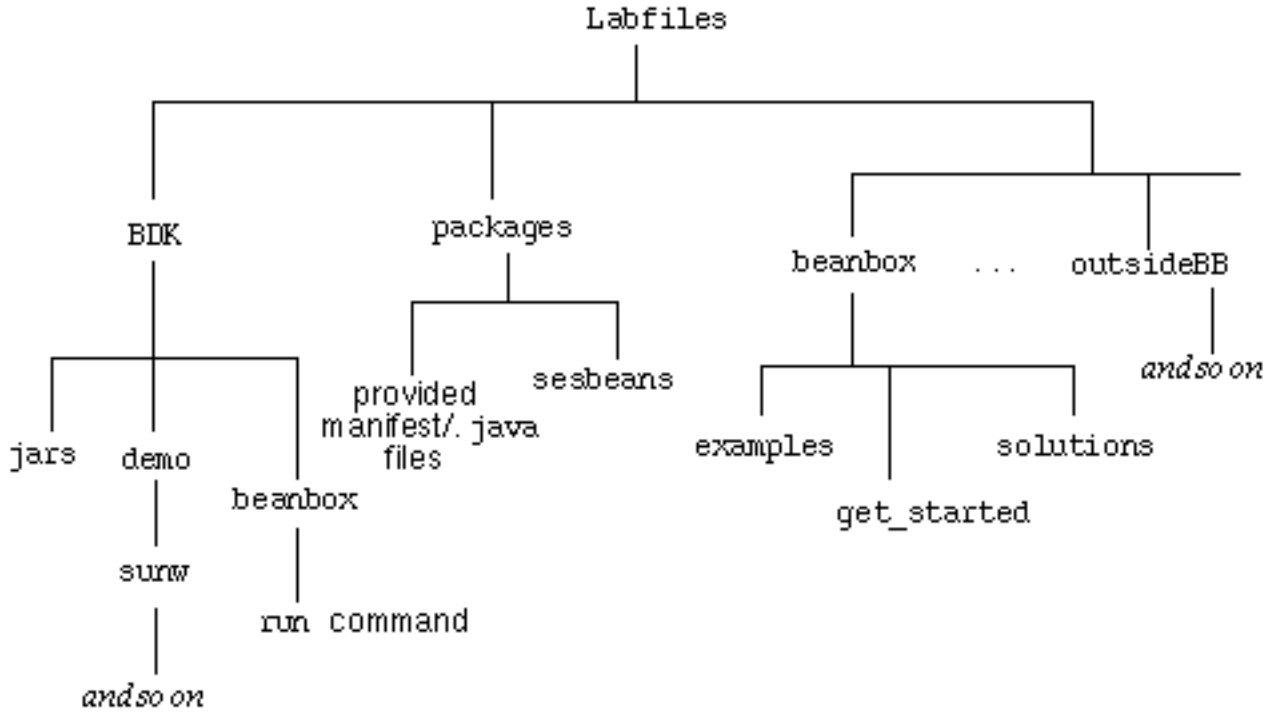
Saving and Restoring the BeanBox

- 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



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

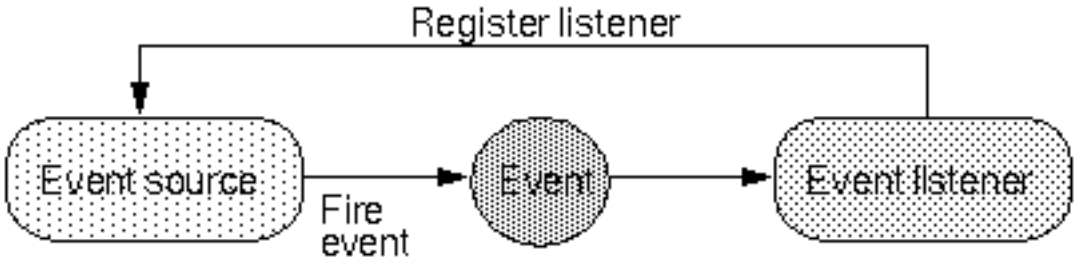
- Objectives
- Relevance
- References

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



Simple Code Example

```
1  import java.awt.*;
2  import java.awt.event.*;
3  public class ButtonHandler implements ActionListener {
4     /**
5      * Component that will contain messages about
6      * events generated.
7      */
8     TextArea output;
9
10    /**
11     * Creates an ActionListener that will put messages in
12     * TextArea everytime event received.
13     */
14    public ButtonHandler(TextArea c) {
15       output = c;
16    }
17
18    /**
19     * When receives action event notification, appends
20     * message to the TextArea passed into the constructor.
21     */
22    public void actionPerformed(ActionEvent e) {
23       output.append("Action occurred:" + e + "\n");
24    }
25 }
26
27 class ActionTester {
28    public static void main(String args[]) {
29       Frame f = new Frame("Button Handler");
30       TextArea area = new TextArea(6, 80);
31       Button button = new Button("Fire Event");
32       button.addActionListener(new ButtonHandler(area));
33       f.add(button, BorderLayout.NORTH);
34       f.add(area, BorderLayout.CENTER);
35       f.pack();
36       f.setVisible(true);
37    }
38 }
```

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 – *XXX*`Listener`
  - ActionListener, ItemListener, and so on
- Event – *XXX*`Event`
  - 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

```
1  package sesbeans.stock;
2  import java.util.*;
3  public class StockPriceChangeEvent extends EventObject {
4     private Stock stock;
5
6     public StockPriceChangeEvent (Object source, Stock s) {
7        super(source);
8        stock = s;
9     }
10 public Stock getStock() {
11       return stock;
12    }
13 }
```

Listeners

- Identify listeners
- Listener interfaces
  - `ActionListener` interface

```
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

```
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
  - `add`*`XXX`*`Listener`
  - `remove`*`XXX`*`Listener`

Multicast Syntax

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

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

public void
   removeStockPriceChangeListener(StockPriceChangeListener spcl) {
   listeners.removeElement(spcl);
}
```

Unicast Syntax

```
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;
   }
}
```

Notifying All Listeners

```
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

- *XXX*`Listener` interface
- *XXX*`Event`
- 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.

Stock Class

```
1  package sesbeans.stock;
2
3  public class Stock {
4     private String company;
5     private String symbol;
6     private double price;
7
8     public Stock(String co, String sym, double p) {
9        company = co;
10       symbol = sym;
11       price = p;
12    }
13
14    public void setCompany(String co) {
15       company = co;
16    }
17
18    public String getCompany() {
19       return company;
20    }
21
22    public void setSymbol(String sym) {
23       symbol = sym;
24    }
25
26    public String getSymbol() {
27       return symbol;
28    }
29
30    public double getPrice() {
31       return price;
32    }
33
34    public void setPrice(double p) {
35       price = p;
36    }
37 }
```

StockPriceChangeEvent Code

```
1  package sesbeans.stock;
2  import java.util.*;
3  public class StockPriceChangeEvent extends EventObject {
4     private Stock stock;
5
6     public StockPriceChangeEvent(Object source, Stock s) {
7        super(source);
8        stock = s;
9     }
10
11    public Stock getStock() {
12       return stock;
13    }
14 }
```

```
1  package sesbeans.stock;
2
3  import java.util.EventListener;
4
5  public interface StockPriceChangeListener extends EventListener {
6     public void priceChange(StockPriceChangeEvent e);
7  }
```

StockWatcher Code

```
1  package sesbeans.stock;
2
3  import java.awt.event.*;
4  import java.util.Vector;
5  import sesbeans.beans.*;
6
7  public class StockWatcher implements ActionListener {
8     Stock stock;
9     Vector listeners = new Vector();
10
11    public StockWatcher() {
12      this(new Stock("Sun Microsystems", "SUNW", 50));
13    }
14
15    public StockWatcher(Stock s) {
16      stock = s;
17
18      //create a timer that generates events every 5 secs,
19      //register this class as interested in timer ticks, and
20      //make timer active.
21      TimerBean t = new TimerBean(5000);
22      t.addActionListener(this);
23      t.setActive(true);
24    }
25
26    public String getCompany() {
27      return stock.getCompany();
28    }
29
30    public void setCompany(String co) {
31      stock.setCompany(co);
32    }
```

StockWatcher Code

```
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
```

StockWatcher Code

```
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 }
```

StockDetail Class

```
1  package sesbeans.stock;
2
3  import java.awt.TextArea;
4
5  public class StockDetail extends TextArea
6     implements StockPriceChangeListener {
7     public StockDetail() {
8        this(null);
9     }
10
11    public StockDetail(Stock s) {
12       super(4, 30);
13       showDetail(s);
14    }
15
16    public void priceChange(StockPriceChangeEvent e) {
17       showDetail(e.getStock());
18    }
19
20    public void showDetail(Stock s) {
21       if (s != null) {
22          setText("Company: " + s.getCompany() + "\n");
23          append("   symbol: " + s.getSymbol() + "\n");
24          append"   price: " + s.getPrice());
25       } else {
26          setText("");
27       }
28    }
29 }
```

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`* `get`*`PropertyName`*`()`
  - `public void set`*`PropertyName`*`(`*`PropertyType`*` a)`
- Boolean properties
  - `public boolean is`*`PropertyName`*`()`
  - `public void set`*`PropertyName`*`(boolean b)`

Naming Conventions for Properties

- Indexed properties

```
public PropertyElement getPropertyName(int index)
public void setPropertyName(int index, PropertyElement element)

public PropertyElement[] getPropertyName()
public void setPropertyName(PropertyElement element[])
```

Naming Conventions for Events

- Multicast event sources

```
public void add EventName Listener( EventName Listener el)

public void remove EventName Listener( EventName Listener el)
```

Naming Conventions for Events

- Unicast event sources

```
public void add<EventName>Listener(<EventName>Listener el)
    throws java.util.TooManyListenersException

public void remove<EventName>Listener(<EventName>Listener 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

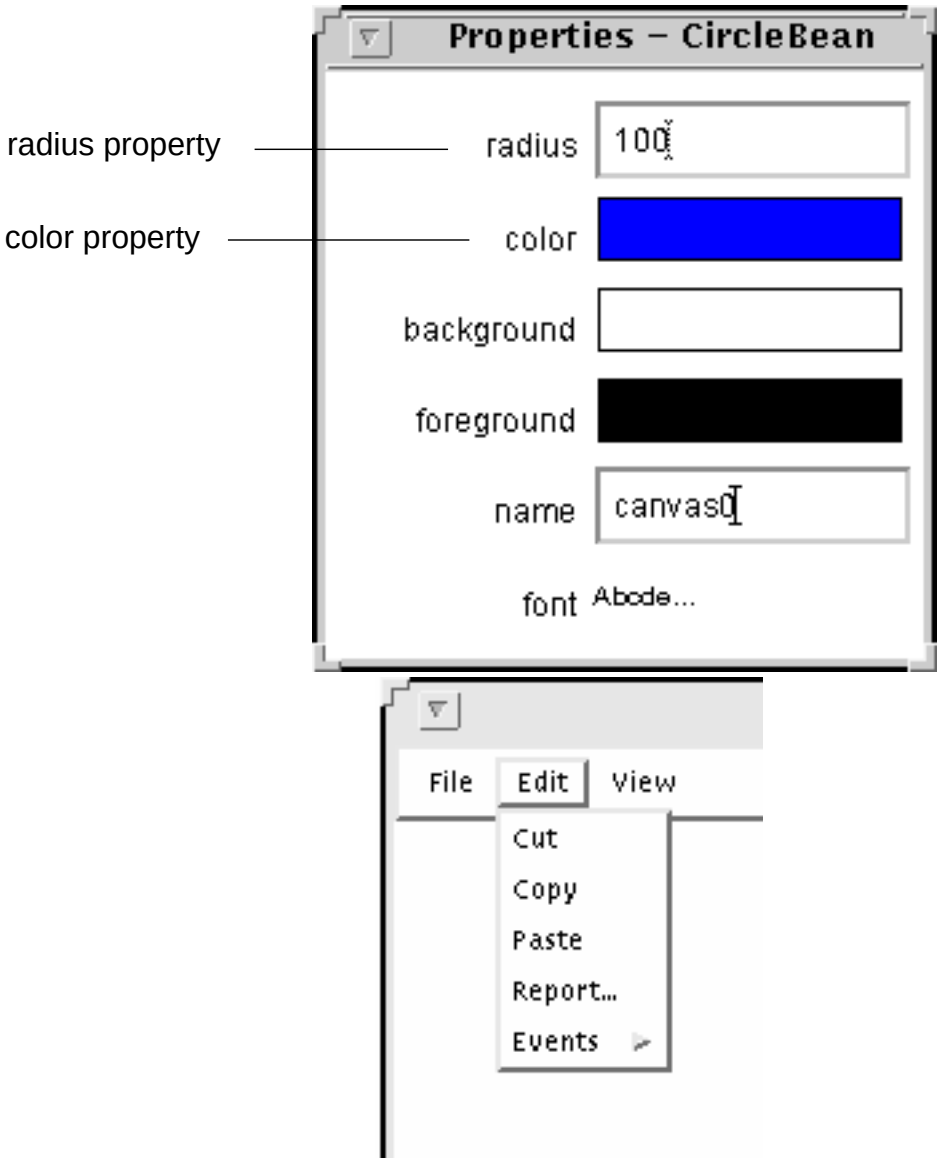
Adding Simple Properties

```
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    }
```

Adding Simple Properties

```
36
37    public int getRadius() {
38       return radius;
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

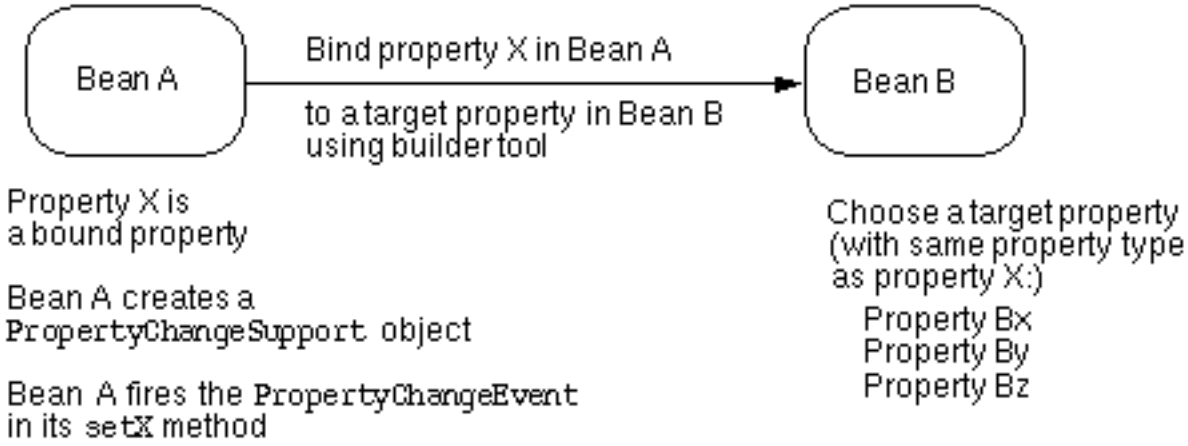


Bound and Constrained Properties

- 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



- `PropertyChangeSupport` class
- Modifying the property `set` method
  - Fire the `PropertyChangeEvent`

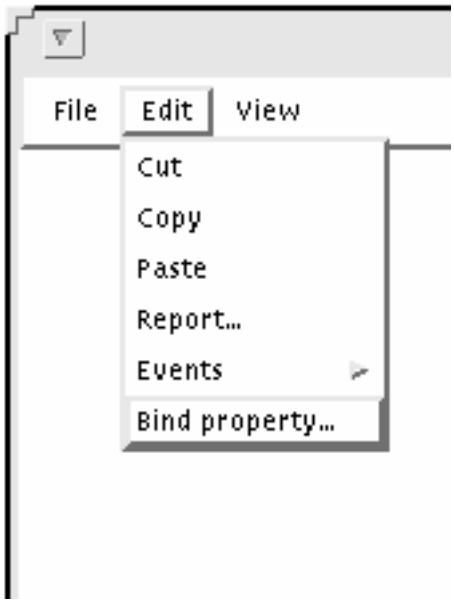
Example of Creating a Bound Property

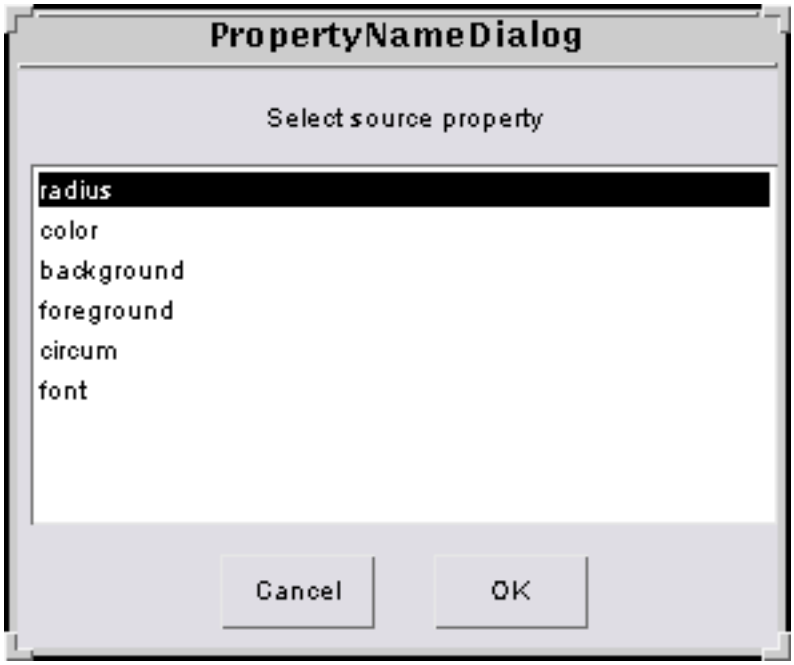
```
15    private PropertyChangeSupport support = new
16         PropertyChangeSupport(this);
17
18    //registration methods for PropertyChangeListeners
19     public void addPropertyChangeLIstener(PropertyChangeListener pcl) {
20       support.addPropertyChangeListener(pc;
21     }
22
23     public 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





Recap of Bound Properties

Code Pieces for Defining Bound Property abc

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

//registration methods for PropertyChangeListeners
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);
}
```

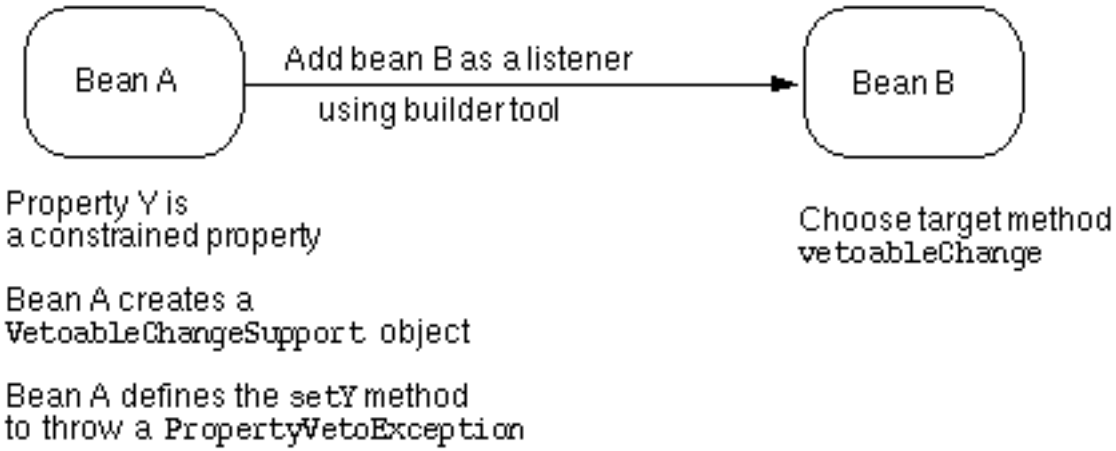
Listener Interface	PropertyChangeListener
Event	Property ChangeEvent
Handler method	propertyChange(PropertyChangeEvent evt)
Utility class	PropertyChangeSupport

Constrained Properties

- Definition
- Overview
  - Order tasks are done in `set`*XXX*

Constrained Properties

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



- Handling vetoes
- `VetoableChangeListener`s

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

```
17   private VetoableChangeSupport vetos = new
18      VetoableChangeSupport(this);


69   public void setRadius(int r) throws PropertyVetoException {
70      int prevRadius = radius;
71      // check for negative values
72      if (r < 0) {
73         System.out.println("Negative radius not allowed!");
74      } else {
75         vetos.fireVetoableChange("radius",
76            prevRadius, r);
77         // no one vetoed, so make the change
78         radius = r;
79         circumference = 2 * 3.14159 * radius;
80         System.out.println("Circumference " + circumference);
81         repaint();
82      }
83   }
```

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

```
private XYZtype xyz;
private VetoableChangeSupport vetoes = new
  VetoableChangeSupport(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;
}
```

Listener interface	`VetoableChangeListener`
Event	`PropertyChangeEvent`
Handler method	`vetoableChange(PropertyChangeEvent evt)` `  throws PropertyVetoException`
Utility class	`VetoableChangeSupport`

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.

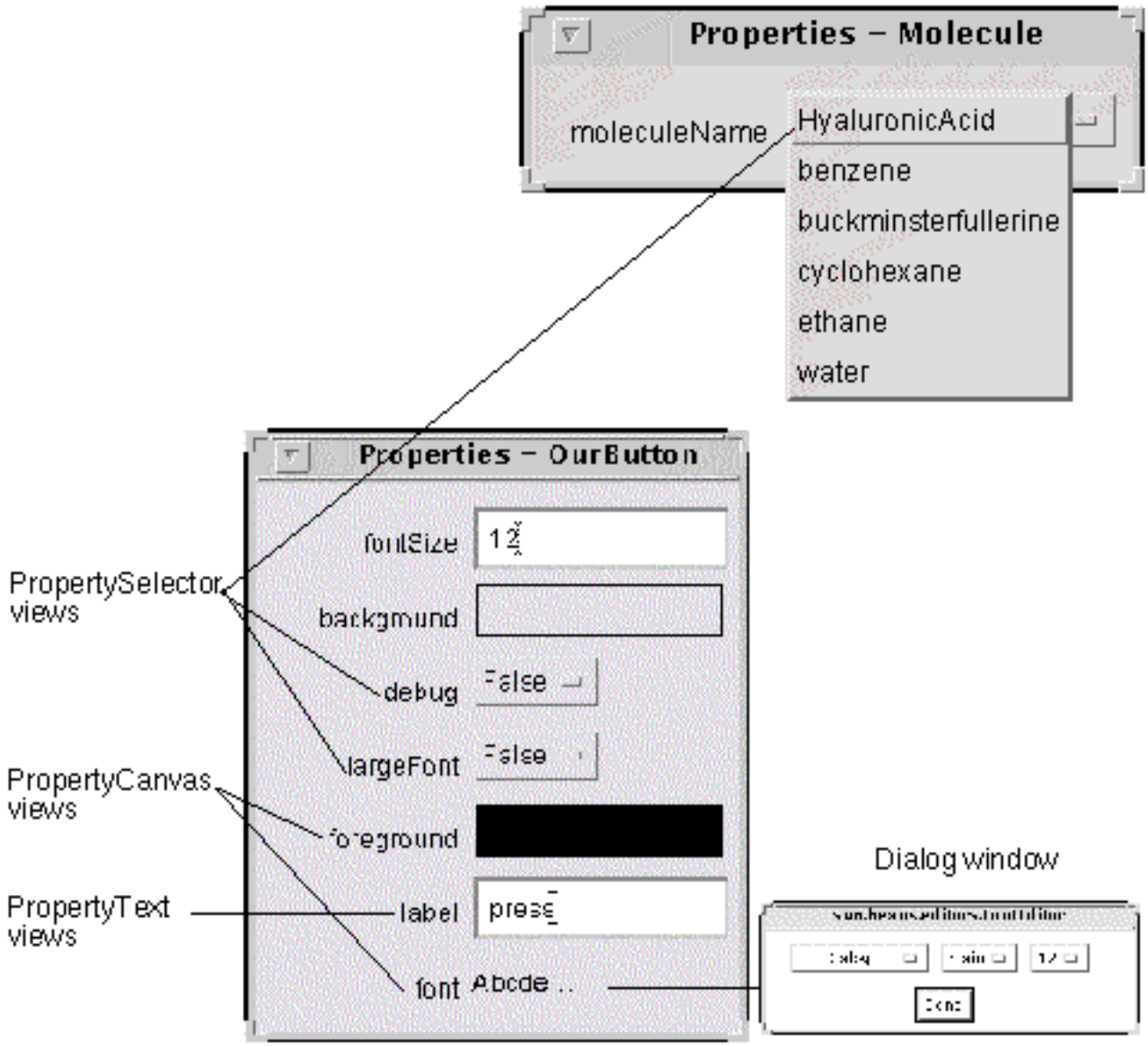
Properties and the BeanBox

- 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



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

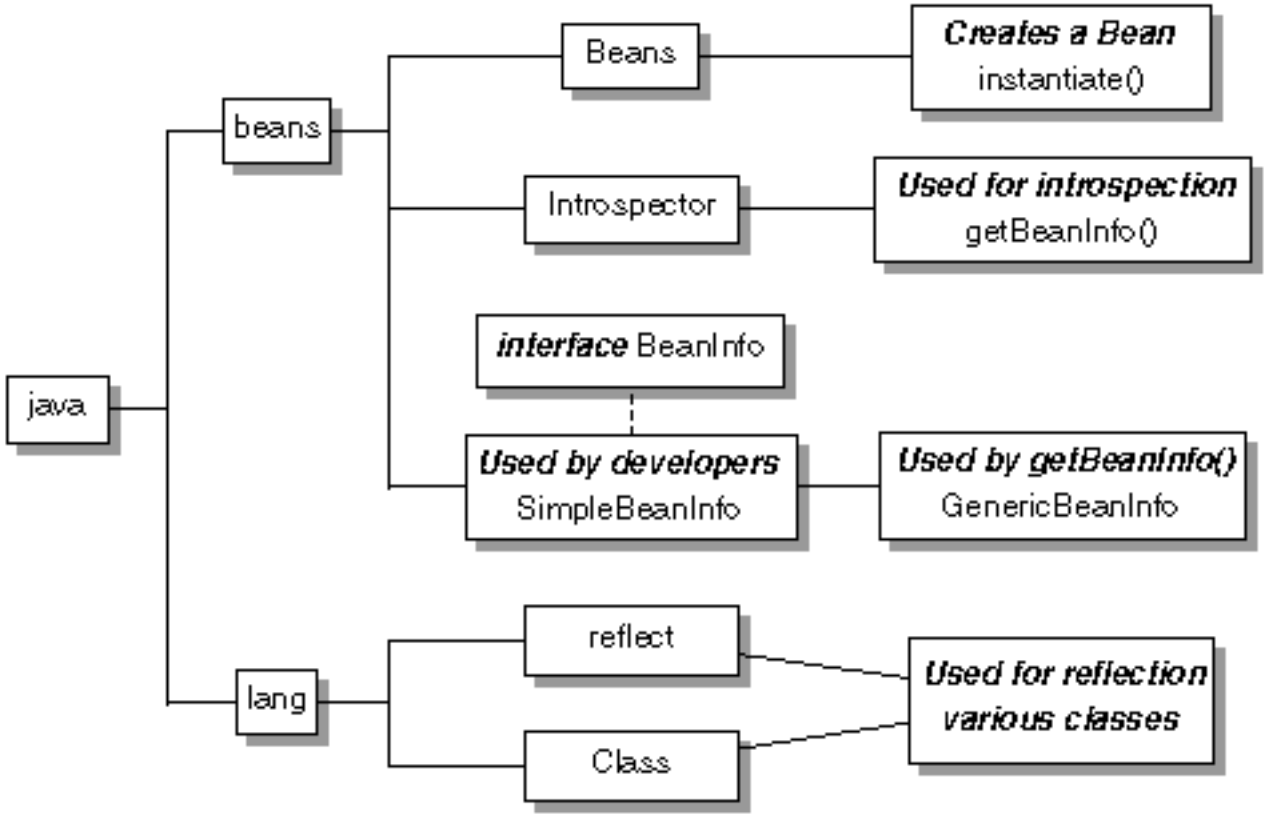
Module Overview

- 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.instantiate` Method

The builder tool instantiates a bean using

```
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

```
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
    cellSize        int                     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.DataTableEvent)
    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

```
1  package sesbeans.circle;
2
3  import java.beans.*;
4
5  public class CircleBeanBeanInfo extends SimpleBeanInfo {
6     public PropertyDescriptor[] getPropertyDescriptors() {
7        try {
8           PropertyDescriptor props[] = {
9              new PropertyDescriptor("radius", CircleBean.class)
10          };
11
12          props[0].setDisplayName("Radius of circle");
13          props[0].setBound(true);
14
15          return props;
16       } catch (IntrospectionException ex) {
17          ex.printStackTrace();
18          return super.getPropertyDescriptors();
19       }
20    }
21 }
```

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

```
1  // Replace CircleBeanBeanInfo.java with the contents of this file
2  package sesbeans.circle;
3
4  import java.beans.*;
5
6  public class CircleBeanBeanInfo extends SimpleBeanInfo {
7      public BeanInfo[] getAdditionalBeanInfo() {
8        return new BeanInfo[] {
9          new CircleBeanAdditionalInfo()
10       };
11     }
12 }
13
14 class CircleBeanAdditionalInfo extends SimpleBeanInfo {
15    public PropertyDescriptor[] getPropertyDescriptors() {
16      try {
17         PropertyDescriptor props[] = {
18            new PropertyDescriptor("radius", CircleBean.class)
19         };
20
21         props[0].setDisplayName("Radius of circle");
22         props[0].setBound(true);
23
24         return props;
25      } catch (IntrospectionException ex) {
26         ex.printStackTrace();
27         return super.getPropertyDescriptors();
28      }
29    }
30 }
```

`BeanInfo` Class That Affects Methods

- Limit number of methods for target bean

```
1  package sesbeans.stock;
2
3  import java.beans.*;
4  import java.lang.reflect.Method;
5
6  public class StockDetailBeanInfo extends SimpleBeanInfo {
7     MethodDescriptor method(String name, Class arg)
8          throws NoSuchMethodException {
9        Method m = StockDetail.class.getMethod(name, new Class[] {arg});
10       return new MethodDescriptor(m);
11    }
12
19    public MethodDescriptor[] getMethodDescriptors() {
20       try {
21          return new MethodDescriptor[] {
22             method("priceChange", StockPriceChangeEvent.class)
23          };
24       } catch(NoSuchMethodException ex) {
25             ex.printStackTrace();
26             return super.getMethodDescriptors();
27       }
28    }
29 }
```

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

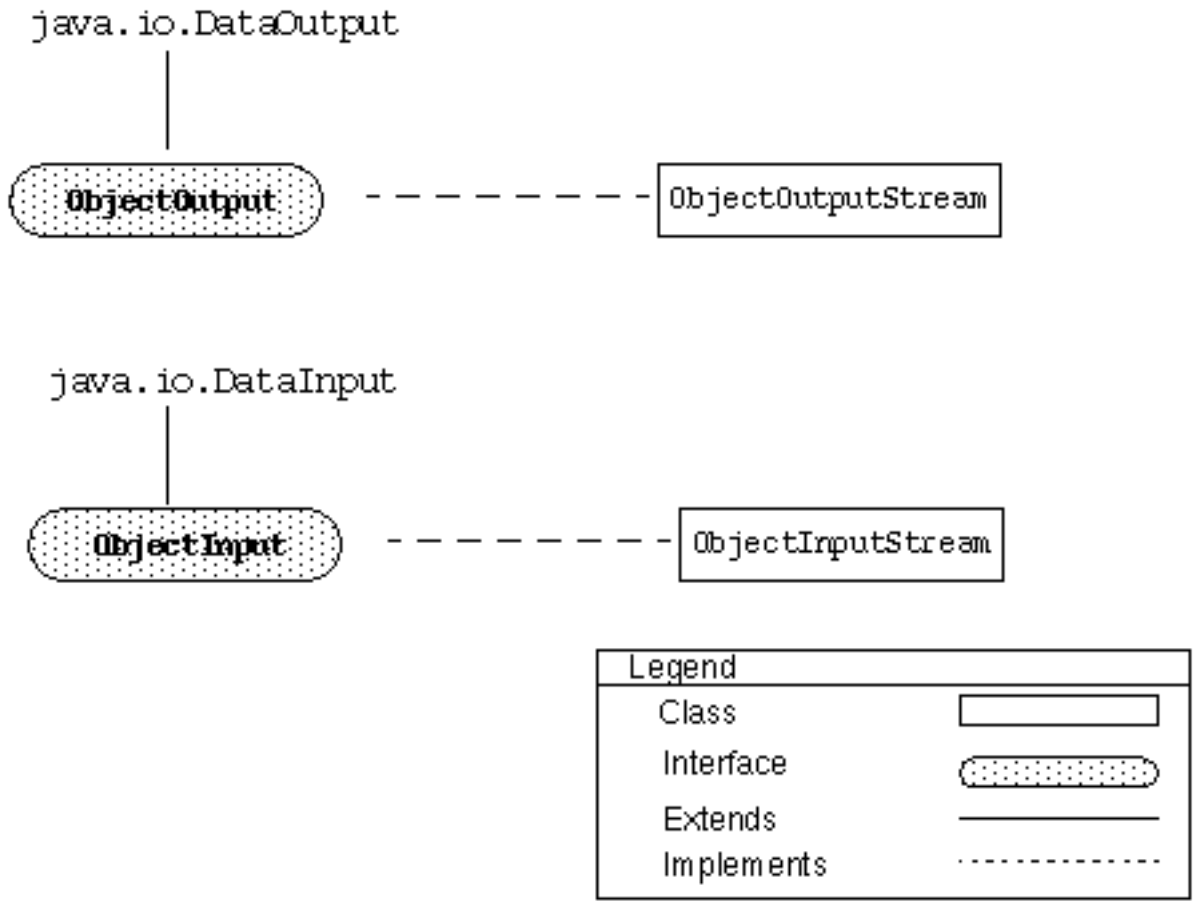
- `Serializable` interface
- Classes that are serialized

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

```
1   public static void main(String args[]) {
2     String serFile = "day.ser";
3
4     try {
5          Date today = new Date();
6
7          FileOutputStream fos = new FileOutputStream(serFile);
8          ObjectOutputStream oos = new ObjectOutputStream(fos);
9          oos.writeObject(today);
10
11         oos.close();
12    }  catch(IOException e) {
13         e.printStackTrace();
14    }
15  }
```

Retrieving Beans From Streams

- `ObjectInputStream` class provides for deserializing persisted objects
- `readObject()` method deserializes objects

```
1   public static void main(String args[]) {
2      try {
3         FileInputStream fis = new FileInputStream("day.ser");
4         ObjectInputStream ois = new ObjectInputStream(fis);
5
6         Date yesterday = (Date)ois.readObject();
7         System.out.println("The date was: "+ yesterday);
8
9         ois.close();
10     } catch(IOException e) {
11        e.printStackTrace();
12     } catch(ClassNotFoundException ex) {
13        //thrown if cannot locate a class used by serialized object
14        ex.printStackTrace();
15     }
16  }
```

`defaultReadObject` and `defaultWriteObject` Methods

- Private methods

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

- Invoking `defaultReadObject` and `defaultWriteObject`

`defaultReadObject` and `defaultWriteObject` Methods

```
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
   istr.defaultReadObject();

   // read back your specific information
}
```

Sample Code

```
1  import java.io.*;
2  import java.awt.Color;
3
4  public class Car implements Serializable {
5
6    private Color bodyColor;
7
8    public Car() {
9      this(Color.black);
10   }
11
12   public Car (Color c) {
13     bodyColor = c;
14   }
15
16   // simple color property
17
18   public Color getColor() {
19     return bodyColor;
20   }
21
22   public void setColor (Color c) {
23     bodyColor = c;
24   }
25
```

Sample Code

```
26  // for each time the car is written, also serialize its green color
27  // component
28
29  private void writeObject(ObjectOutputStream oos) {
30    try {
31
32      // use the default writing mechanism
33      oos.defaultWriteObject();
34
35      // write anything else you need to serialize
36      oos.writeInt(bodyColor.getGreen());
37
38      System.out.println("car written: " + bodyColor);
39    } catch(Exception e) {}
40  }
41
42  // for each deserialization, also read the int that was written
43
44  private void readObject(ObjectInputStream ois) {
45    int i=0;
46
47    try {
48
49      // restore the serialized data values for this object
50      ois.defaultReadObject();
51
52      // also read other information that was serialized manually
53      i = ois.readInt();
54
55      System.out.println("car read: " + bodyColor);
56    } catch(Exception e) {}
57  }
58 }
```

Sample Code

```
1  import java.awt.Color;
2  import java.io.*;
3  import java.beans.*;
4
5  public class SerlCar {
6
7    private Car accord;
8    private FileOutputStream fos;
9    private ObjectOutputStream oos;
10   private FileInputStream fis;
11   private ObjectInputStream ois;
12
13   public SerlCar() {
14     accord = new Car();
15     accord.setColor(Color.green);
16   }
17
18   public void writeCar() {
19
20     try {
21       fos = new FileOutputStream("Car.ser");
22       oos = new ObjectOutputStream(fos);
23
24       oos.writeObject(accord);
25
26       oos.close();
27     } catch (IOException e) { }
28
29   }
30
```

Sample Code

```
31 public void readCar() {
32
33     try {
34       fis = new FileInputStream("Car.ser");
35       ois = new ObjectInputStream(fis);
36
37       accord = (Car)ois.readObject();
38
39       ois.close();
40     } catch (Exception excep){
41       System.out.println(excep);
42     }
43   }
44
45   public void getBean() {
46     ClassLoader cl = SerlCar.class.getClassLoader();
47     Car myCar = null;
48
49     try {
50       myCar = (Car)Beans.instantiate(cl, "Car");
51     } catch (Exception excep) {
52       System.out.println(excep);
53     }
54      System.out.println("deserialized bean: " + myCar.getColor());
55   }
56
57   public static void main(String[] args) {
58     SerlCar sc = new SerlCar();
59     sc.writeCar();
60     sc.readCar();
61     sc.getBean();
62   }
63 }
```

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

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

Reading from a serialized stream

```
String serFile = "filename.ser";
try {
      FileInputStream fileIStream = new FileInputStream(serFile);
      ObjectInputStream is = new ObjectInputStream(fileIStream);
      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

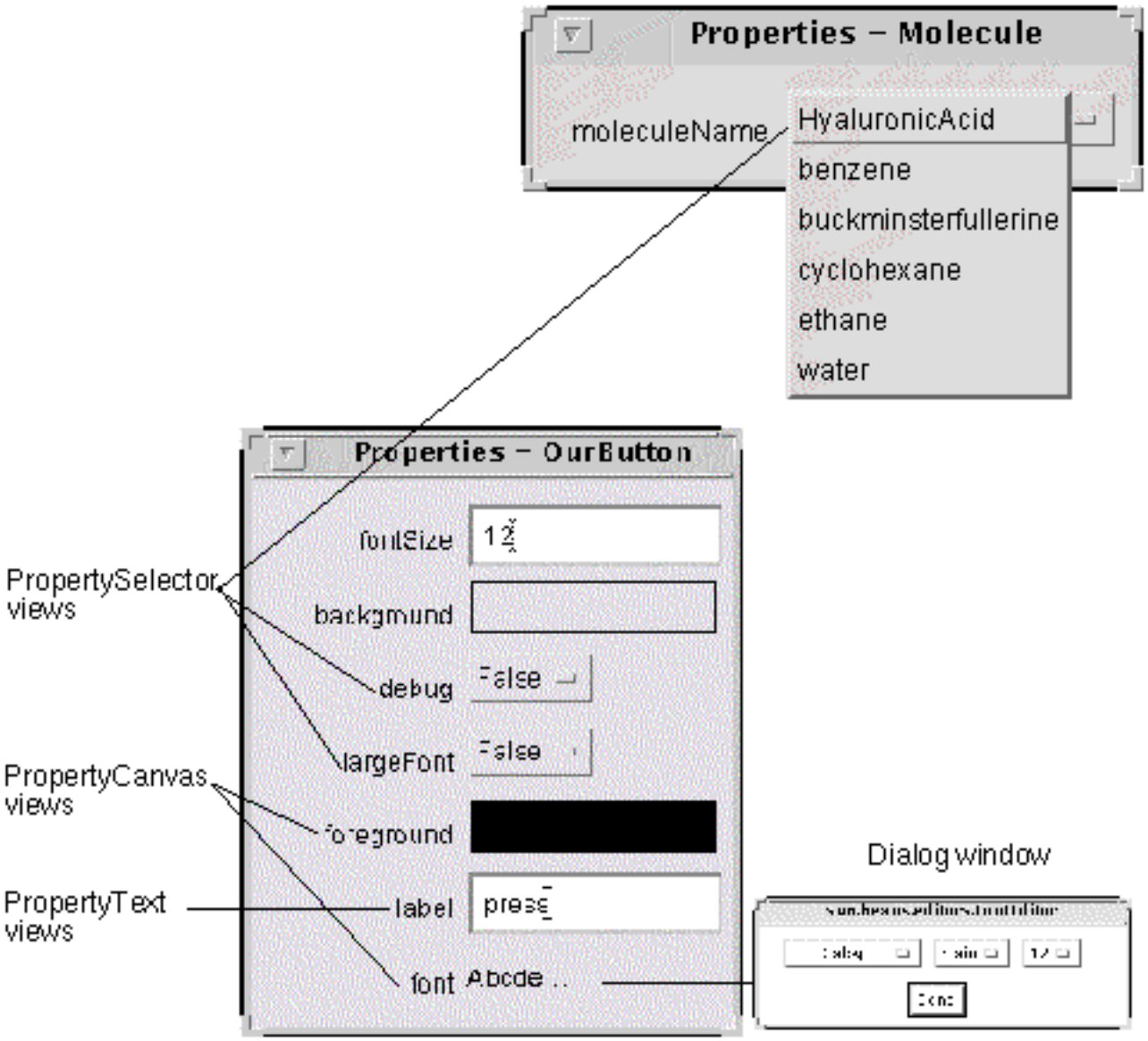
Property Sheets and Property Editors

- 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



Property Editor Basics

- Description of the property editor API
  - `PropertyEditor` interface
  - `PropertyEditorSupport` class

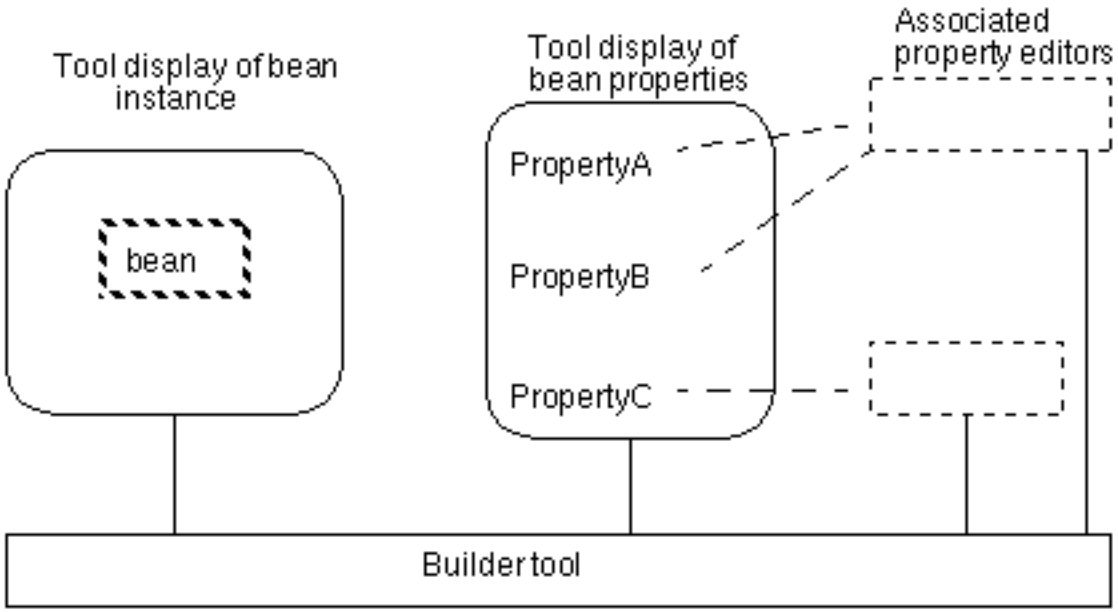
Behavior Requirements

- Support one of three display styles
- Support `setValue`
- Support `PropertyChangeListeners`
- 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

Multiple Line Label

```
1  package sesbeans.labelEditor;
2
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.beans.*;
6
7  public class LabelEditor implements PropertyEditor {
8     private String value;
9
10    //Property editors should provide a default constructor.
11    public LabelEditor() { }
12
13    // Maintain the property value for the property being edited.
14    public void setValue(Object valFromBT) {
15    value = (String)valFromBT;
16    }
17
18    //Return the value of the property.
19    public Object getValue() {
20    return value;
21    }
      ...
86    PropertyChangeSupport listeners = new
87          PropertyChangeSupport(this);
88
89    //Register listeners for the PropertyChangeEvent.
90    public void addPropertyChangeListener(PropertyChangeListener pl)
{
91    listeners.addPropertyChangeListener(pl);
92    }
93
94    //Remove listeners for the PropertyChangeEvent.
95    public void removePropertyChangeListener
96       (PropertyChangeListener pl){
97    listeners.removePropertyChangeListener(pl);
98    }
99 }
```

Custom GUI

- Methods required
  - `isPaintable`
  - `paintValue`
  - `supportsCustomEditor`
  - `getCustomEditor`

LabelEditor Custom GUI

```
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    }
27
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    }
50
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    }
56
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    }
```

LabelEditor Custom GUI

```
64
65    //Return a custom GUI component that is used to edit the
66    // property value.
67    public Component getCustomEditor() {
68    final TextArea labelArea = new TextArea(value);
69    labelArea.setSize(300, 150);
70    labelArea.addTextListener(new TextListener(){
71          public void textValueChanged(TextEvent e) {
72               value = labelArea.getText();
73             // Notify listeners that the value for this property
74             // has been modified.
75               listeners.firePropertyChange(null, null, null);
76          }
77    } );
78    return labelArea;
79    }
80
81    //Return true if the class provides a custom editor.
82    public boolean supportsCustomEditor() {
83    return true;
84    }
```

Choice of Tags

- `getTags` returns a non-null `String[]` of choices
- `BoolEditor` is an example of this type of editor

BoolEditor Choice of Tags Methods

```
1  package sun.beans.editors;
2
3   // Property editor for a java built-in "boolean" type.
4
5  import java.beans.*;
6  public class BoolEditor extends PropertyEditorSupport {
7
8     public String getJavaInitializationString() {
9     // This must return local independent Java.
10    if (((Boolean)getValue()).booleanValue()) {
11       return ("true");
12    } else {
13       return ("false");
14    }
15    }
16    public String getAsText() {
17    if (((Boolean)getValue()).booleanValue()) {
18       return ("True");
19    } else {
20       return ("False");
21    }
22    }
23    public void setAsText(String text) {
24    if (text.toLowerCase().equals("true")) {
25       setValue(Boolean.TRUE);
26    } else if (text.toLowerCase().equals("false")) {
27       setValue(Boolean.FALSE);
28    } else {
29       throw new java.lang.IllegalArgumentException(text);
30    }
31    }
32    public String[] getTags() {
33    String result[] = { "True", "False" };
34    return result;
35    }
36 }
```

Simple String in a Text Field

- The least complex display style
- Return non-null `String` from `getAsText`
- Support `setAsText`

`StringEditor` From the BeanBox

```
1  package sun.beans.editors;
2
3  import java.beans.*;
4
5  public class StringEditor extends PropertyEditorSupport {
6
7     public String getJavaInitializationString() {
8     return "\"" + getValue() + "\"";
9     }
10
11    public void setAsText(String text) {
12    setValue(text);
13    }
14 }
```

Making Your Property Editor Known

- `PropertyEditorManager` class
- Naming conventions
- `BeanInfo` file

BeanInfo for Multiple Line Label Bean

```
1  package sesbeans.beans;
2
3  import java.beans.*;
4  import java.awt.*;
5
6  public class MultilineLabelBeanInfo extends SimpleBeanInfo {

11    PropertyDescriptor property(String name, String desc)
12       throws IntrospectionException {
13    PropertyDescriptor p = new PropertyDescriptor(name,
14             MultilineLabel.class);
15    p.setShortDescription(desc);
16    return p;
17    }
18
19    public PropertyDescriptor[] getPropertyDescriptors() {
20    try {
21       PropertyDescriptor props[] = {
22          property("label", "The contents of the control"),
23          property("eolStyle",
24             "The method used to determine end of lines"),

32    };
33
34       props[0].setPropertyEditorClass(LabelEditor.class);
35       props[1].setPropertyEditorClass(EolStyleEditor.class);
36       props[2].setPropertyEditorClass(AlignmentStyleEditor.class);
37       return props;
38    } catch(IntrospectionException ex) {
39       ex.printStackTrace();
40       return super.getPropertyDescriptors();
41    }
42    }

47 }
```

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:

```
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 `PropertyChangeListeners`
- Defining `setObject()`

Defining BeanInfo

```
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

```
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...
}
```

Adding and Removing `PropertyChangeListeners`

```
// 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

```
1  package sunw.demo.buttons;
2
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.beans.*;
6
7  public class ExplicitButtonCustomizer extends Panel implements
8        Customizer, KeyListener {
9
10    public ExplicitButtonCustomizer() {
11       setLayout(null);
12    }
13
14    public void setObject(Object obj) {
15       target = (ExplicitButton) obj;
16
17       Label t1 = new Label("Caption:", Label.RIGHT);
18       add(t1);
19       t1.setBounds(10, 5, 60, 30);
20
21       labelField = new TextField(target.getLabel(), 20);
22       add(labelField);
23       labelField.setBounds(80, 5, 100, 30);
24
25       labelField.addKeyListener(this);
26    }
27
28    public Dimension getPreferredSize() {
29       return new Dimension(200, 40);
30    }
31
32    /**
33    * @deprecated provided for backward compatibility with old layout
34       managers.
35    */
36    public Dimension preferredSize() {
37       return getPreferredSize();
38    }
39
```

Example of a Customizer

```
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

```
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

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

Provide a null argument constructor for the customizer

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

Support the addition and removal of a PropertyChangeListener

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

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

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

Define the setObject(Object bean) method

```
private MyBean bean;
public void setObject(Object beanToCustomize) {
  bean = (MyBean) beanToCustomize;
  // 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

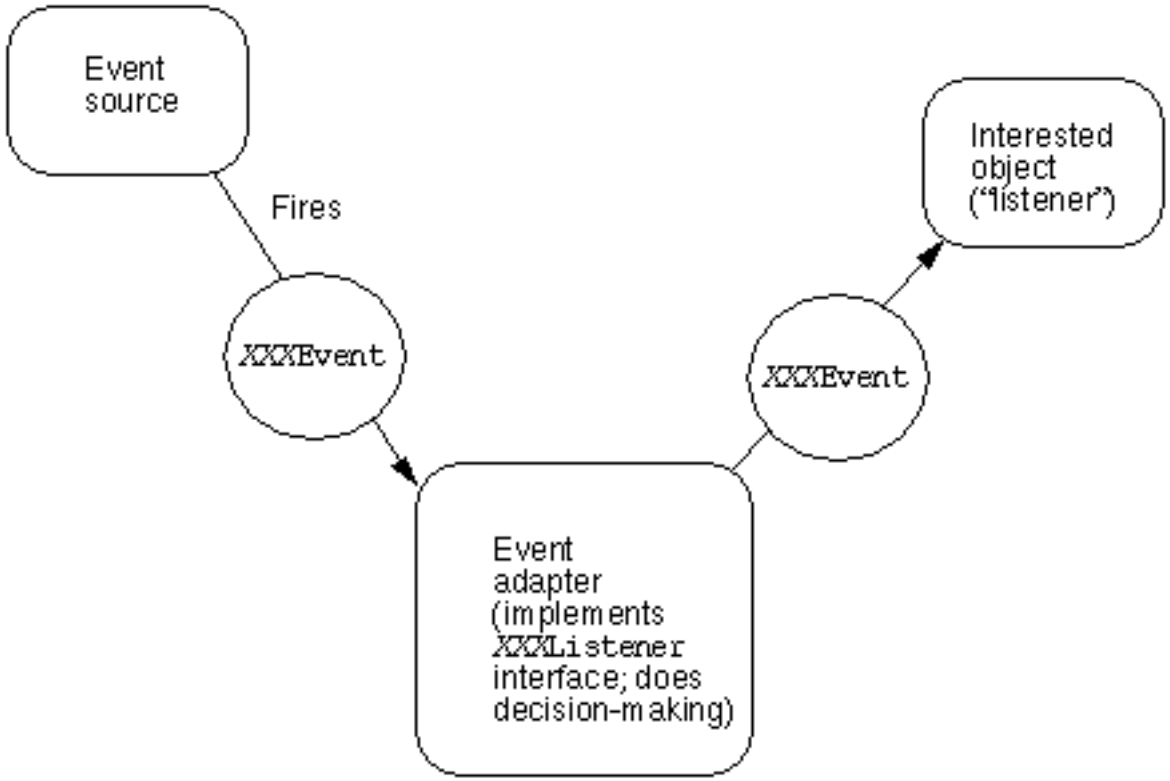
Event Adapters

Module Overview

- Objectives
- Relevance
- References

What Is an Event Adapter?

- Definition
- Diagram overview



Adapters Used in the BeanBox

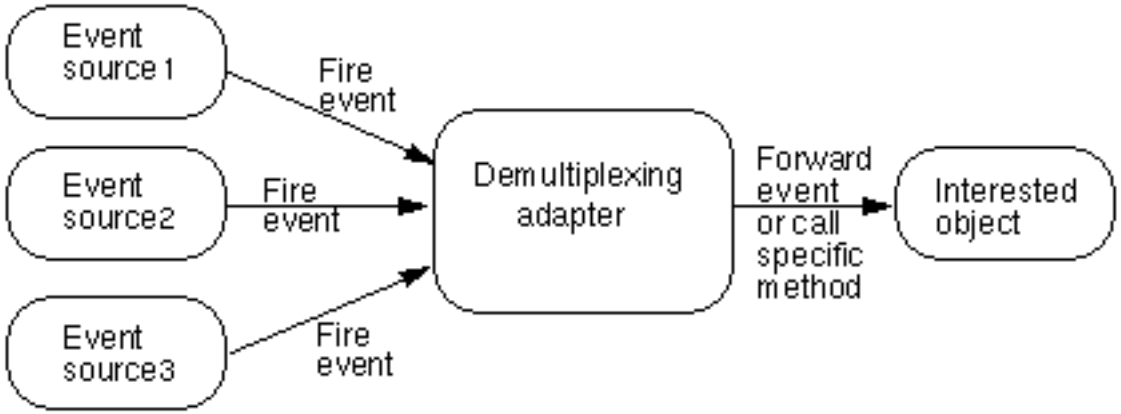
- When are hookup classes created?
- Where is the generated code stored?
- Example of an adapter hookup class:

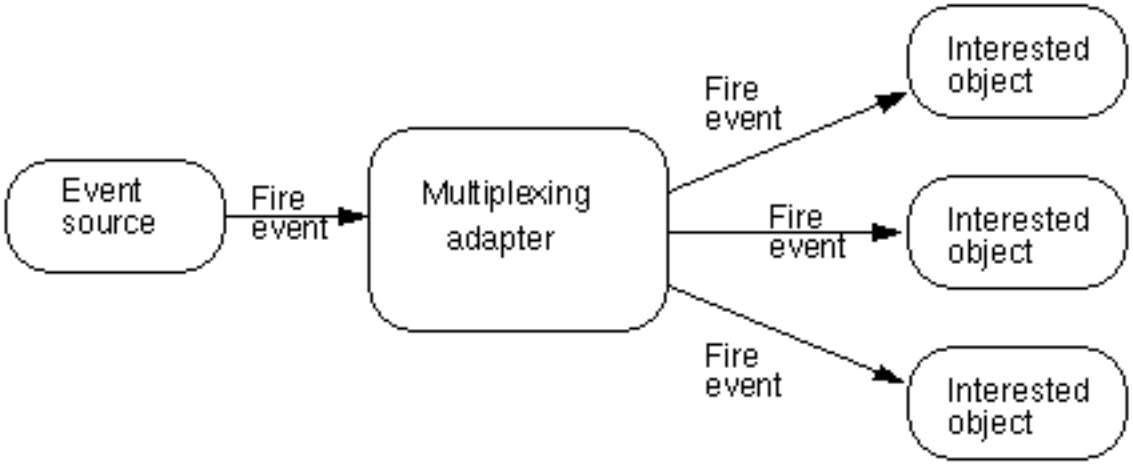
```
package tmp.sun.beanbox;

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



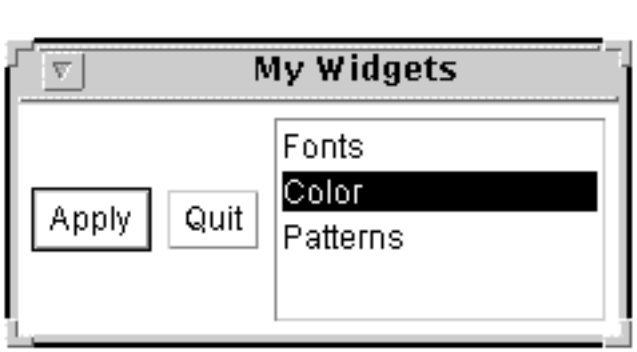


Differentiating Adapters From Normal Listeners

- Case with no adapters
- Case with adapters

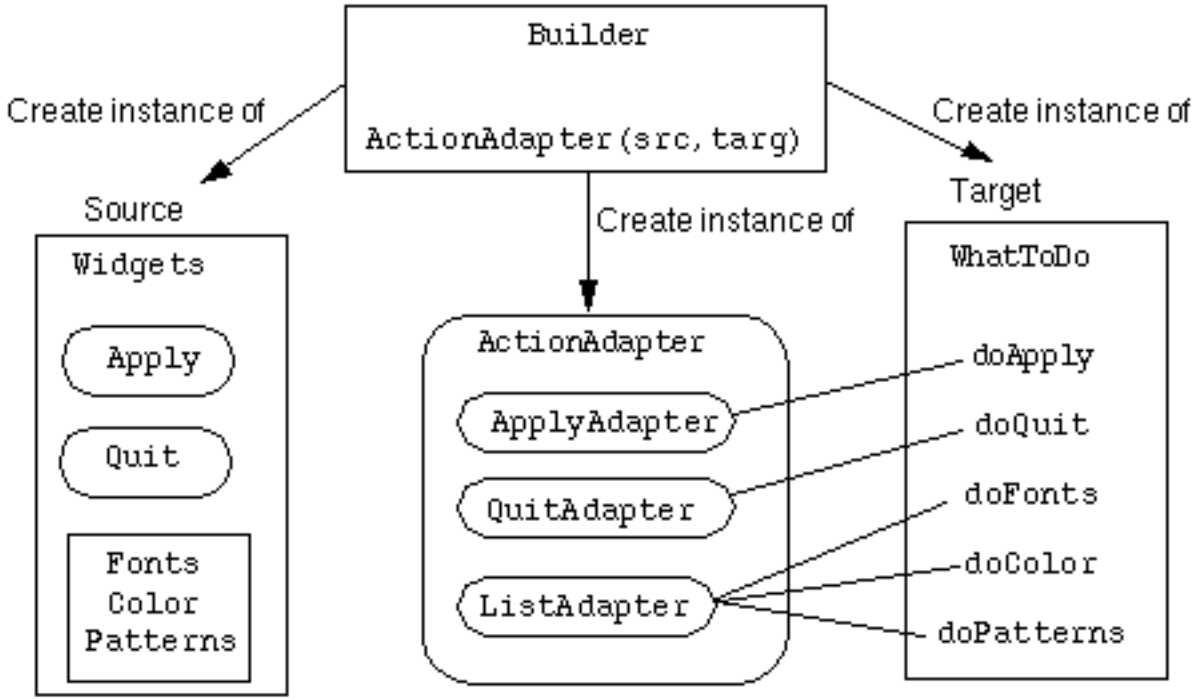
Demultiplexing Adapter Example

- Application example



- Description of application
  - `Builder.java`
  - `Widgets.java`
  - `ActionAdapter.java`
  - `WhatToDo.java`

Description of Application



Builder.java Code

```
1  package sesbeans.widadapter;
2
3  public class Builder {
4
5      // The builder is responsible for attaching the source to the
6      // adapter to the target.
7
8     public static void main(String[] args) {
9        Widgets src = new Widgets();
10       WhatToDo targ = new WhatToDo();
11
12       // To connect the pieces, make the adapter which knows about
13       // both the source and the target
14       ActionAdapter adapter = new ActionAdapter(src, targ);
15    }
16 }
```

Widgets.java Code

```
1  package sesbeans.widadapter;
2
3  import java.awt.*;
4  import java.awt.event.*;
5
6  // Create the Widgets for this application
7  // Give any listener the opportunity to choose
8  // any or all of the widgets to listen to.
9
10 public class Widgets {
11
12    private Frame fr;
13    private Button apply;
14    private Button quit;
15    private List myList;
16
17    public Widgets() {
18
19       // Make the GUI with its widgets

33       fr.add(apply);
34       fr.add(quit);
35       fr.add(myList);
36
37       // Put up the frame
38       fr.pack();
39       fr.setVisible(true);
40    }
41
42    public void addApplyListener(ActionListener targ) {
43       apply.addActionListener(targ);
44    }
45
46    public void removeApplyListener(ActionListener targ) {
47       apply.removeActionListener(targ);
48    }
```

Widgets.java Code

```
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 }
```

ActionAdapter.java Code

```
1  package sesbeans.widadapter;
2
3  import java.awt.*;
4  import java.awt.event.*;
5
6  // An adapter that receives ActionEvents from widgets in
7  // the source bean. Not only does the adapter call
8  // appropriate methods in the target, it can do any other
9  // processing needed; in this case, it simply shows or
10 // hides a frame when one of the source buttons is pushed.
11
12  public class ActionAdapter {
13
14    private Widgets src;
15    private WhatToDo targ;
16
17    public ActionAdapter(Widgets s, WhatToDo t) {
18       src = s;
19       targ = t;
20
21       // We're choosing to make a different subtype of adapter
22       // for each source. That didn't have to be the case, but
23       // it does make for cleaner actionPerformed() methods.
24       src.addApplyListener(new ApplyAdapter());
25       src.addQuitListener(new QuitAdapter());
26       src.addListListener(new ListAdapter());
27    }
28
29    class ApplyAdapter implements ActionListener {
30       public void actionPerformed(ActionEvent evt) {
31          // Since it came from the Apply in the source,
32          // do something interesting like displaying an icon
33          displayIcon();
34          // Call target's appropriate method; could have passed
35          // information, had it been needed
36          targ.apply();
37       }
38     }
```

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 }
```

WhatToDo.java Code

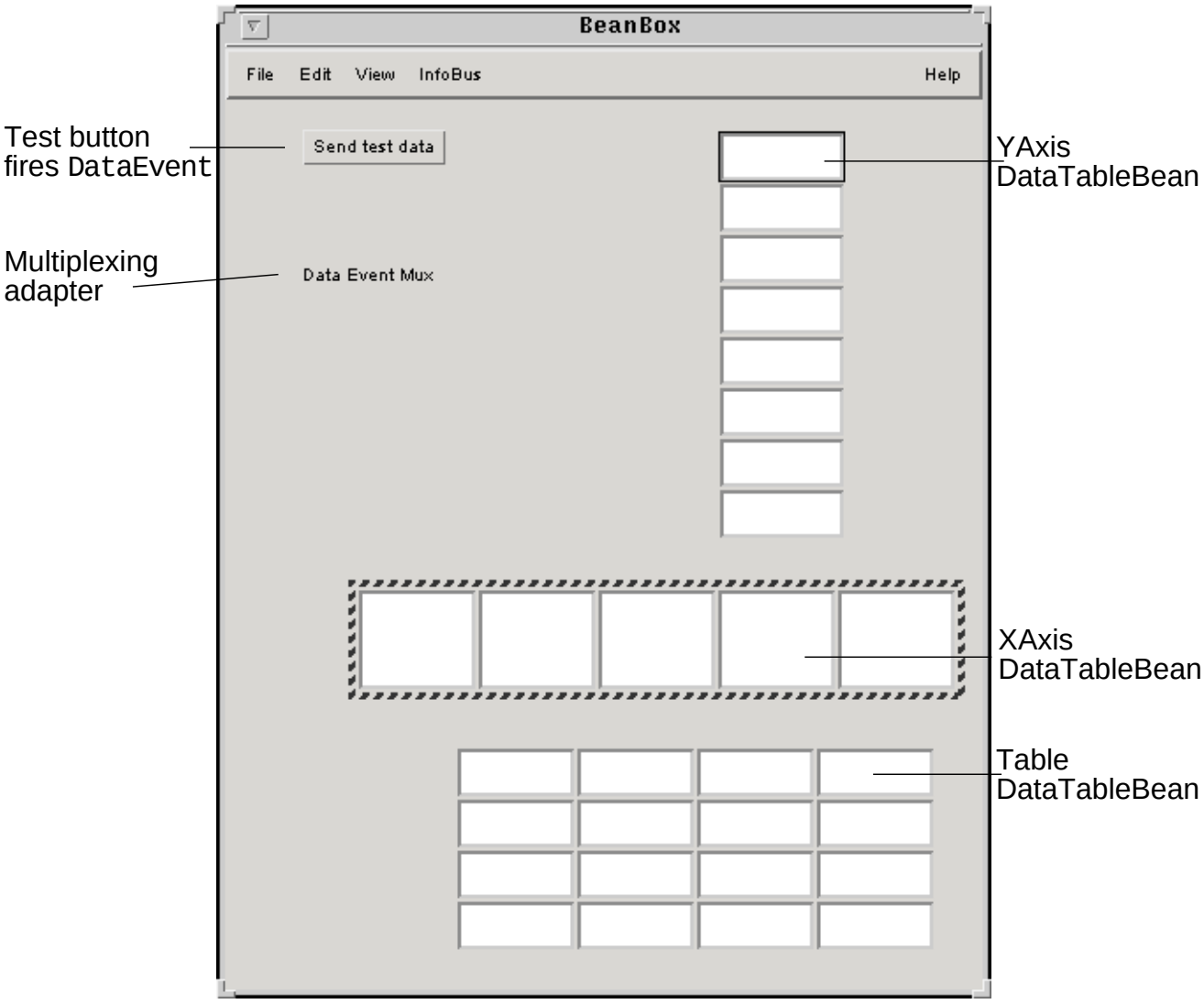
```
1  package sesbeans.widadapter;
2
3  import java.awt.*;
4
5  // The target has several jobs that it can do
6
7  public class WhatToDo {
8
9     public void apply() {
10       System.out.println("Do apply ...");
11    }
12
13    public void quit() {
14       System.out.println("Quitting...");
15    }
16
17    public void doFonts() {
18       System.out.println("Put up Fonts dialog");
19    }
20
21    public void doColor() {
22       System.out.println("Put up Color dialog");
23    }
24
25    public void doPatterns() {
26       System.out.println("Put up Patterns dialog");
27    }
28 }
```

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

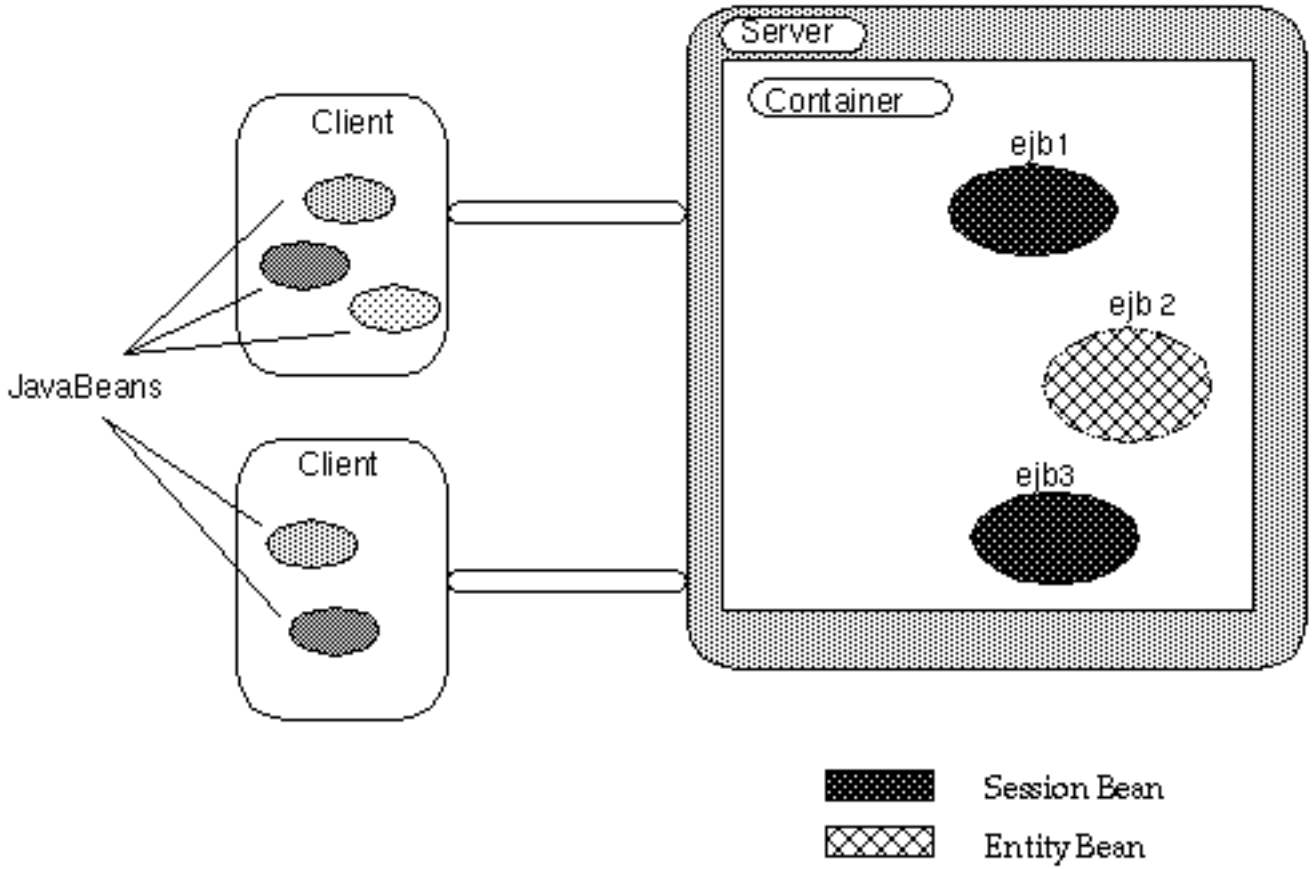
EJB Features

- 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

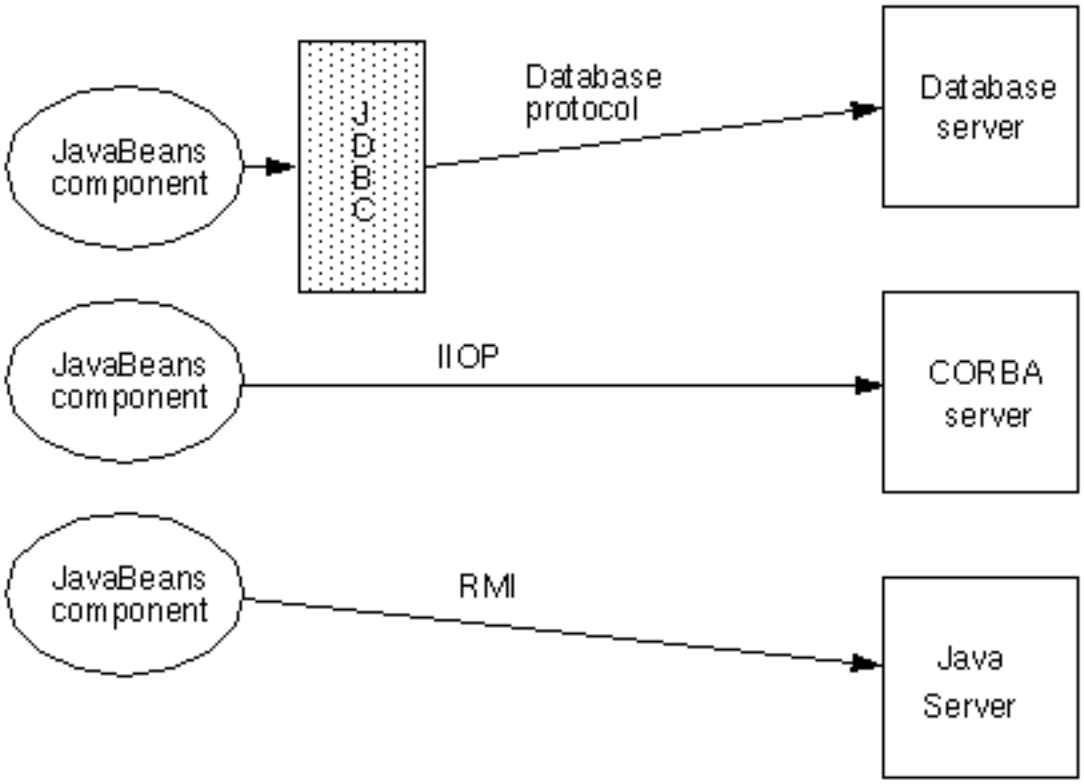
JavaBeans	Enterprise JavaBeans
Visual and deployed on client-side, usually	Non-visual and deployed on server-side, always
Deployed like any class – as an applet or application	Deployed in a container
Properties and behaviors usually introspected by builder tool	Properties and context discovered from deployment descriptor file
An unmanaged component	A component managed by the container and server
Events used extensively	Events not used, normally

Applications for Distributed Beans

- Workflow applications
- Application servers
- Agents

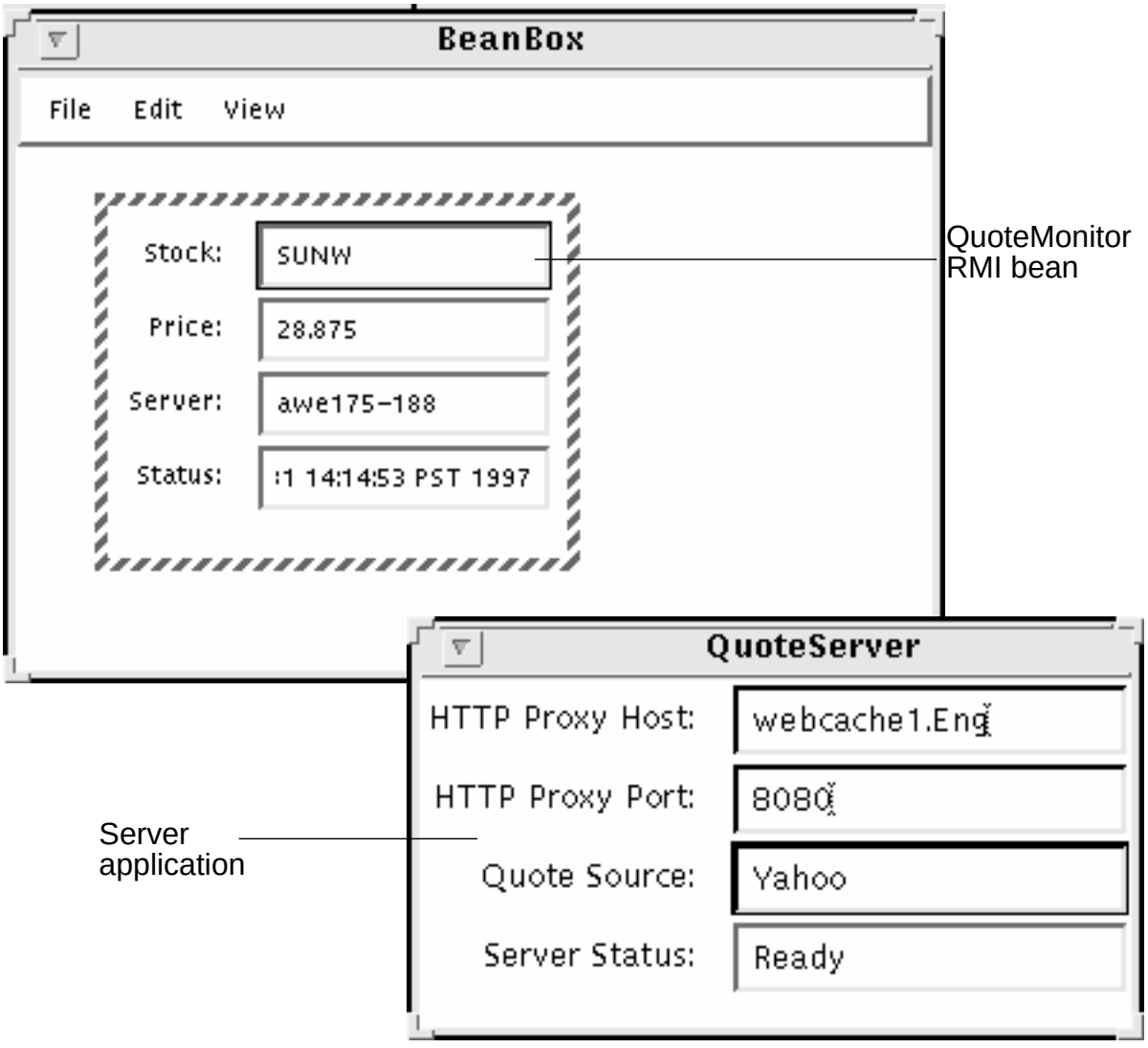
Distributed Computing Technologies

- JDBC
- RMI
- JavaIDL



BeanBox RMI Bean

- QuoteMonitorBean

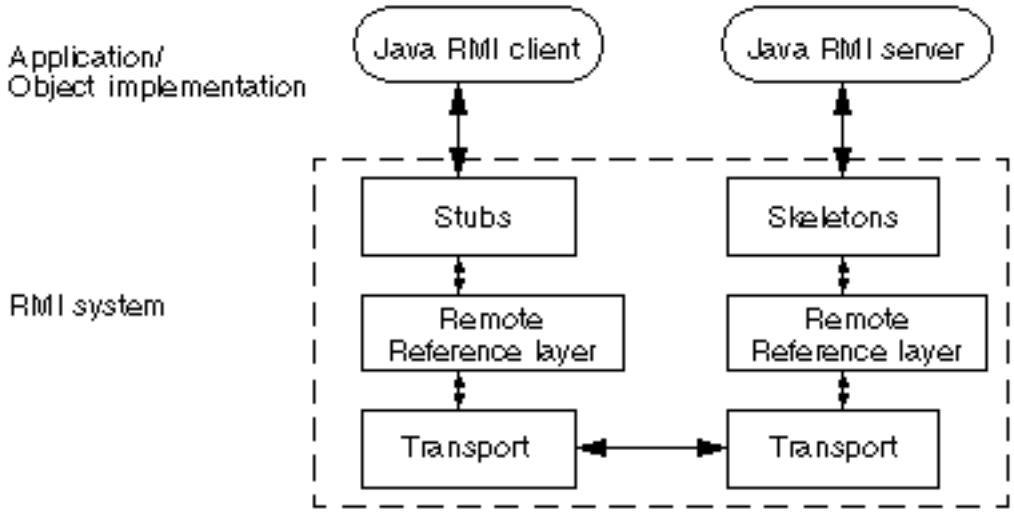


Definition of RMI

- 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

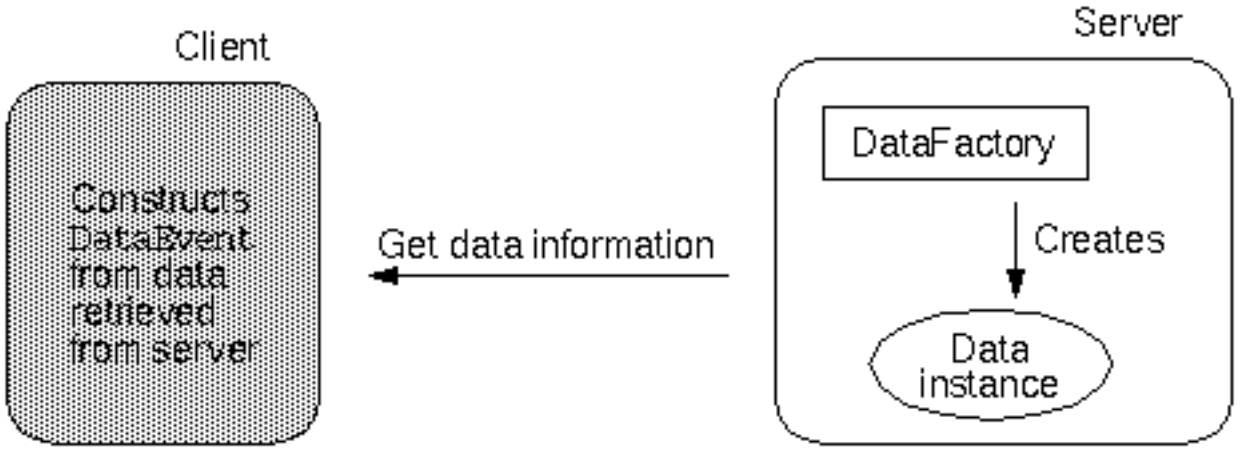


RMI Architecture Overview

- Transport layer
- Remote Reference layer
- RMI stubs and skeletons
- `rmic` command
- `rmiregistry` application

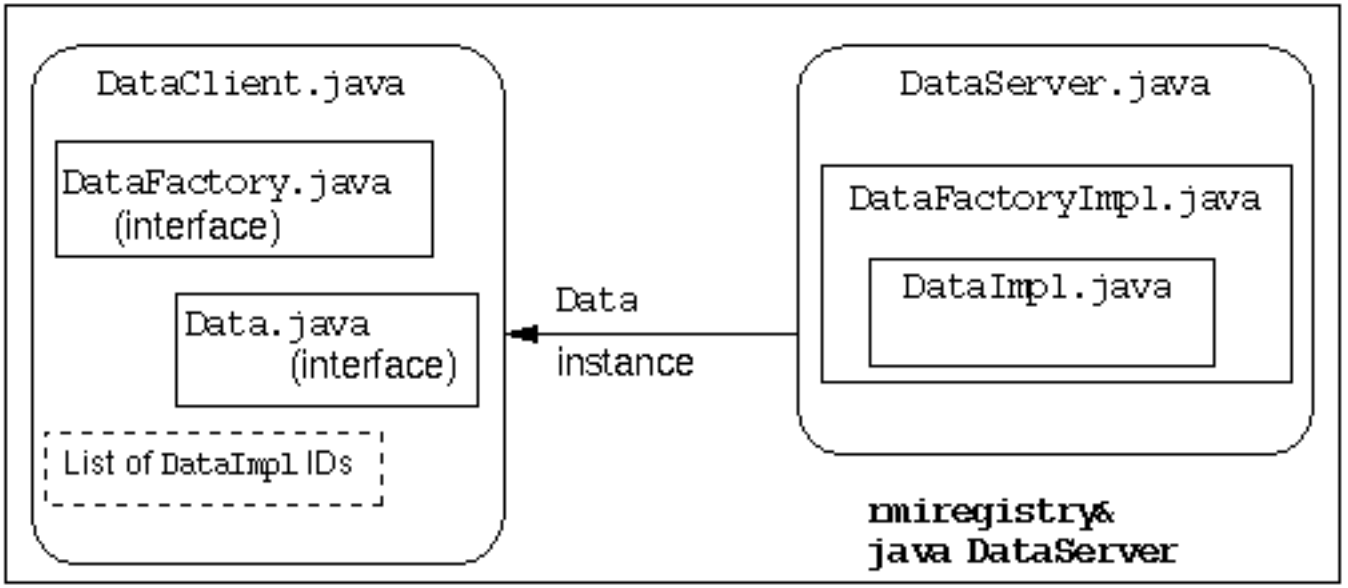
RMI Exercise Code

- 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:



Data.java File

```
1  package sesbeans.rmilab;
2
3  import java.rmi.*;
4  import java.awt.Dimension;
5
6  public interface Data extends Remote{
7
8     public String getID() throws RemoteException;
9
10    public String getXLabel() throws RemoteException;
11
12    public String getYLabel() throws RemoteException;
13
14    public String [] getXAxis() throws RemoteException;
15
16    public String [] getYAxis() throws RemoteException;
17
18    public Dimension getSize() throws RemoteException;
19
20    public String [] [] getDataValues() throws RemoteException;
21 }
```

DataFactory.java File

```
1  package sesbeans.rmilab;
2
3  import java.rmi.*;
4
5  public interface DataFactory extends Remote{
6
7     public Data getData(String id) throws RemoteException;
8     public String [] getDataList() throws RemoteException;
9     public void addData(DataImpl dImpl) throws RemoteException;
10 }
```

DataImpl.java File

```
1  package sesbeans.rmilab;
2
3  import java.rmi.*;
4  import java.rmi.server.*;
5  import java.awt.Dimension;
6
7  public class DataImpl extends UnicastRemoteObject implements Data{
8
9     private String id, xLabel, yLabel;
10    private String [] xAxis, yAxis;
11    private String [][] dataValues;
12    private Dimension size;
13
14    public DataImpl(String id, String xLabel, String yLabel)
15    throws RemoteException {
16
17        this.id = id;
18        this.xLabel = xLabel;
19        this.yLabel = yLabel;
20        this.size = new Dimension (8,5);
21        xAxis = new String [size.height];
22        yAxis = new String [size.width];
23        dataValues = new String [yAxis.length] [xAxis.length];
24    }
25
26    public String getID() throws RemoteException {
27        return id;
28    }
```

DataImpl.java File

```
29 public String getXLabel() throws RemoteException {
30       return xLabel;
31   }
32
33   public String getYLabel() throws RemoteException {
34       return yLabel;
35   }
36
37   public String [] getXAxis() throws RemoteException {
38       return xAxis;
39   }
40
41   public String [] getYAxis() throws RemoteException {
42       return yAxis;
43   }
44
45   public Dimension getSize() throws RemoteException {
46       return size;
47   }
48
49   public String [] [] getDataValues() throws RemoteException {
50       return dataValues;
51   }
52
53   public void setXAxis(String [] sa1) {
54       xAxis = sa1;
55   }
56
57   public void setYAxis(String [] sa2) {
58       yAxis = sa2;
59   }
60
61   public void setSize(int width, int height) {
62       size = new Dimension(width, height);
63   }
64
65   public void setDataValues(String [] [] sa3) {
66       dataValues = sa3;
67   }
68 }
```

DataFactoryImpl.java

```
1  package sesbeans.rmilab;
2
3  import java.rmi.*;
4  import java.rmi.server.*;
5  import java.util.Vector;
6
7  public class DataFactoryImpl extends UnicastRemoteObject implements
8     DataFactory{
9
10    private Vector storage;
11
12    public DataFactoryImpl() throws RemoteException {
13       storage = new Vector(1,1);
14    }
15
16    public Data getData(String id) throws RemoteException {
17       for (int i = 0; i < storage.size(); i++) {
18
19       if (((DataImpl)storage.elementAt(i)).getID().equals(id)){
20          return (Data)storage.elementAt(i);
21       }
22    }
23    return null;
24 }
25    public String [] getDataList() throws RemoteException {
26       String [] names = new String [storage.size()];
27
28       for (int i = 0; i < storage.size(); i++) {
29          names[i] = ((DataImpl)(storage.elementAt(i))).getID();
30        }
31       return names;
32    }
33
34    public void addData(DataImpl dImpl) throws RemoteException {
35       storage.addElement(dImpl);
36    }
37
38 }
```

Server Code Overview

- Description of the DataServer application
- Description of binding a `DataFactoryImpl` object to a name in `rmiregistry`

```
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`

```
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

```
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

```
64 public BeanDescriptor getBeanDescriptor() {
65    BeanDescriptor back = new BeanDescriptor(beanClass, customizerClass);
66    back.setValue("hidden-state", Boolean.TRUE);
67    return back;
68 }
69
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

```
42 public EventSetDescriptor[] getEventSetDescriptors() {
43 try {
44      EventSetDescriptor push = new EventSetDescriptor(beanClass,
45         "actionPerformed",
46         java.awt.event.ActionListener.class,
47         "actionPerformed");
48
49      EventSetDescriptor changed = new EventSetDescriptor(beanClass,
50         "propertyChange",
51         java.beans.PropertyChangeListener.class,
52         "propertyChange";
53
54      push.setDisplayName("button push");
55      changed.setDisplayName("bound property change");
56
57      EventSetDescriptor[] rv = { push, changed};
58      return rv;
59   } catch (IntrospectionException e) {
60      throw new Error(e.toString());
61   }
62 }
```

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

```
1  import java.beans.*;
2  import sunw.demo.molecule.*;
3  import java.io.*;
4
5  // Create the serialized file of molecule changed to Benzene
6  public class MyMolSer {
7     private static Molecule moleculeB;
8
9     public MyMolSer() {
10       ClassLoader cl = null;
11       try {
12          cl = MyMolSer.class.getClassLoader();
13          moleculeB = (Molecule)
14             Beans.instantiate(cl,"sunw.demo.molecule.Molecule");
15          moleculeB.setMoleculeName("benzene");
16          FileOutputStream fos = new
17             FileOutputStream("sunw/demo/molecule/
moleculeSerFile.ser");
18          ObjectOutputStream outputStream = new
ObjectOutputStream(fos);
19          outputStream.writeObject(moleculeB);
20       } catch ( Exception e) {
21          throw new Error(e.toString());
22       }
23    }
24
25    public static void main (String args[]) {
26       MyMolSer m = new MyMolSer();
27    }
28 }
```

java MyMolSer

Restoring the Bean

```
1   import java.awt.Frame;
2   import java.awt.Component;
3   import java.beans.Beans;
4   import java.io.*;
5
6   // Read the serialized file and display the bean
7   public class BeanUnSer extends Frame {
8      private static Component myBean;
9      public BeanUnSer(String beanSerFile) {
10        super("Unserialized from: "+beanSerFile);
11        try {
12           myBean = (Component) Beans.instantiate(null,beanSerFile);
13        } catch ( Exception e) {
14           System.out.println("Error unserializing bean " +
15              beanSerFile+": "+e);
16        }
17        this.add(myBean);
18        this.setSize(300,300);
19        this.show();
20     }
21
22     public static void main (String args[]) {
23        BeanUnSer m = new BeanUnSer(args[0]);
24     }
25  }
```

`java BeanUnSer sunw.demo.molecule.moleculeSerFile`

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

```
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:widadpater.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       }
```

Sample Code

```
27
28      // From the URL, obtain a class loader
29      urlCL = new URLClassLoader(regURL);
30
31      // Get a class from the jar
32      try {
33         classFromJar = urlCL.loadClass("sesbeans.widadapter.Builder");
34         try {
35            // Prove that everything worked by forcing the loaded
36            // class to be used
37            // Could also do
38            // Object o =(Object)Beans.instantiate(urlCL,
39            //      "sesbeans.widadapter.Builder");
40            Object o = classFromJar.newInstance();
41            System.out.println(o);
42         } catch (Exception e) {
43            System.out.println("can't make new instance: " + e);
44         }
45      } catch (ClassNotFoundException e) {
46         System.out.println("bad class load: " + e);
47      }
48
49      // Get a data file from the jar
50      URL gifURL = classFromJar.getResource("dukeJB.gif");
51   }
52
53 }
```

Sample Code

- Instantiating a bean in an applet

```
<APPLET
   ARCHIVE="twosimple.jar"
   CODE="LabelChanger"
   WIDTH=300
   HEIGHT=100
></APPLET>
```

```
1  import sesbeans.twoSimpleBeans;
2
3  import java.awt.Frame;
4  import java.awt.BorderLayout;
5  import java.applet.Applet;
6  import java.beans.*;
7  import java.io.*;
8
9  public class LabelChanger extends Applet {
10
11    public void init() {
12       MyButton myButton = null;
13       MyText myText = null;
14       try {
15          ClassLoader cl = this.getClass().getClassLoader();
16          myText = (MyText) Beans.instantiate(cl,
17             "sesbeans.twoSimpleBeans.MyText");
18          myButton = (MyButton) Beans.instantiate(cl,
19             "sesbeans.twoSimpleBeans.MyButton");
20       } catch (Exception e) {
21          throw new RuntimeException(e.toString());
22       }
```

Sample Code

```
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 -jar jarname.jar
```

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

```
Main-Class: LabelChanger
```

Hooking Beans Together

- Instantiate the beans using `Beans.instantiate`
- Register listener beans with source beans

  `sourceBean.add`*`XXX`*`Listener(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

```
1  import java.beans.*;
2  import java.awt.*;
3  import sunw.demo.juggler.*;
4
5  public class TestApplet {
6     private Frame fr;
7     private Juggler jugg;
8
9     public static void main (String args[]){
10       TestApplet tst = new TestApplet();
11       tst.doDisplay();
12    }
13
14    public void doDisplay() {
15       ClassLoader cl;
16
17       cl = TestApplet.class.getClassLoader();
18       try {
19          jugg= (Juggler)Beans.instantiate(cl,
20             "sunw.demo.juggler.Juggler");
21       } catch (Exception e) {
22          throw new Error(e.toString());
23    }
24       fr = new Frame("test applet instantiation");
25       fr.add(jugg);
26       jugg.start();
27       fr.pack();
28       fr.setVisible(true);
29    }
30 }
```

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

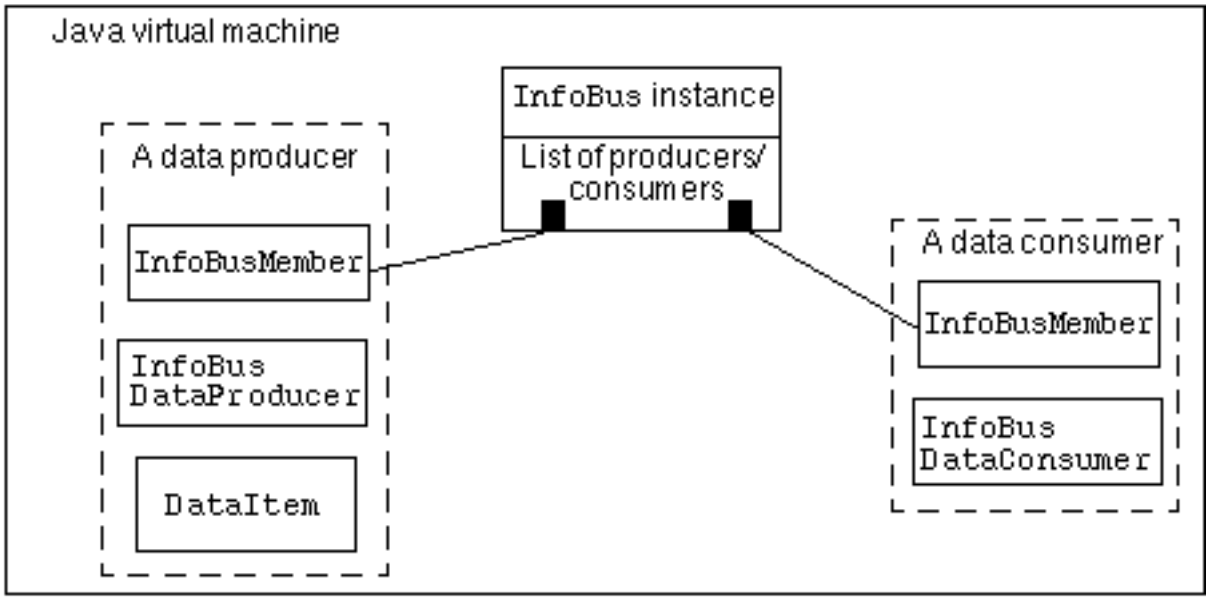
Handling or Sharing Data Among Beans

- 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
```

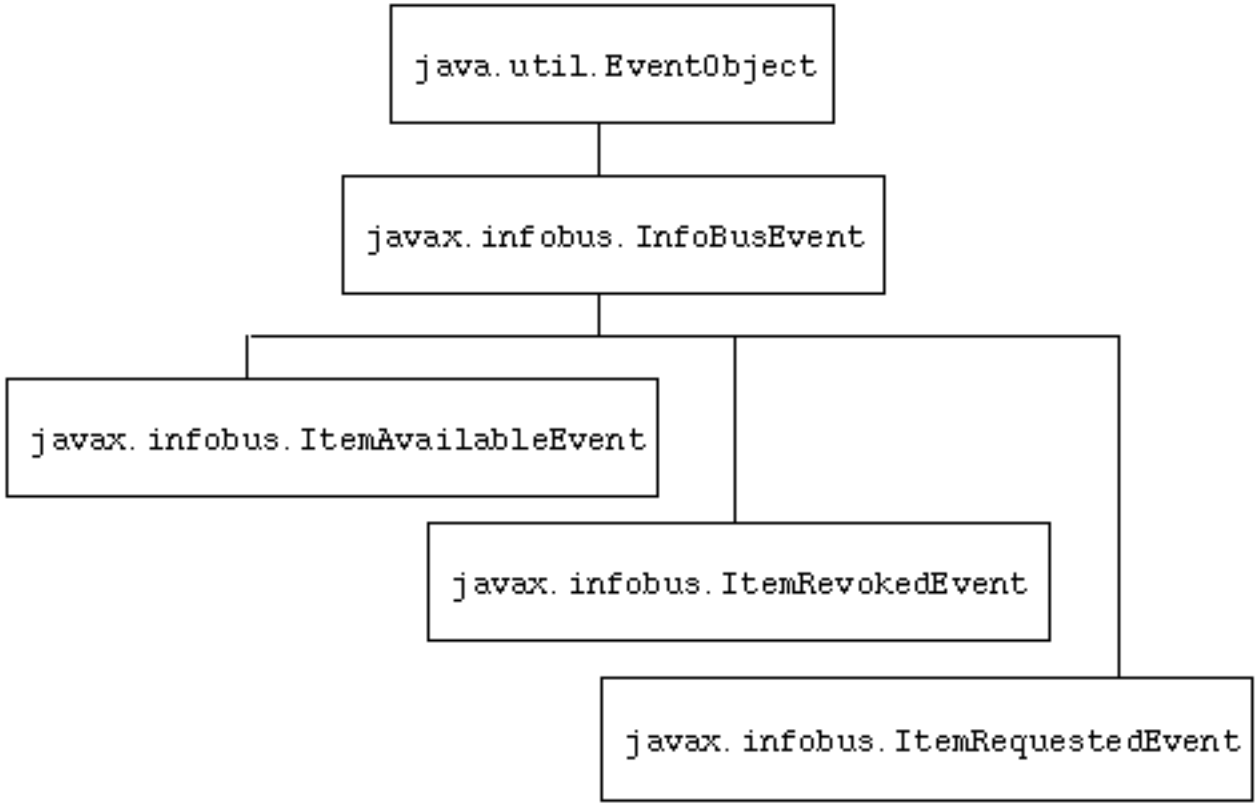
and

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

and

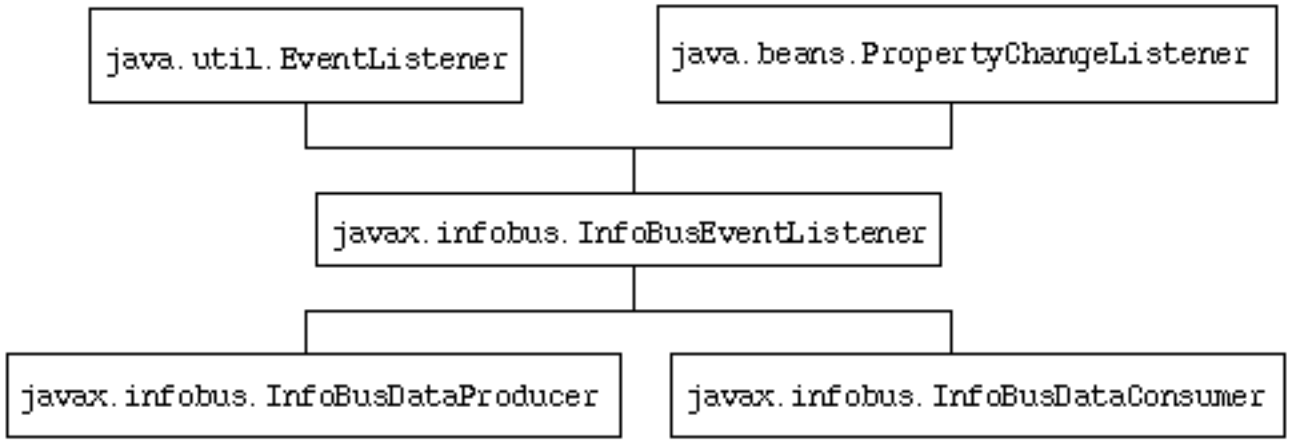
```
public class SampleConsumer extends Applet
  implements InfoBusMember,InfoBusDataProducer,
     InfoBusDataConsumer,DataItemChangeListener,
     ActionListener, ItemListener
```

InfoBus Events



InfoBus Event Listeners

- `InfoBusItemAvailableEvent`
- `InfoBusItemRevokedEvent`
- `InfoBusItemRequestedEvent`
- Event handlers



Events, Firing Methods, and Handlers

Events	InfoBus Methods	Consumer Methods	Producer Methods
`InfoBusItemAvailableEvent`	`fireItemAvailable`	`dataItemAvailable`	
`InfoBusItemRevokedEvent`	`fireItemRevoked`	`dataItemRevoked`	
`InfoBusRequestedEvent`	`findDataItem` `findMultipleDataItems`		`dataItemRequested`

InfoBus `DataItem` Interface

- Interface definition

```
public abstract interface DataItem
```

- Access interfaces
  - `ImmediateAccess`
  - `ArrayAccess`
  - `RowsetAccess`
  - `ScrollableRowsetAccess`
  - `DbAccess`

Sample `DataItem`s From the InfoBus Software

```
public class MonetaryDataItem implements
ImmediateAccess, DataItem, DataItemChangeManager
```

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

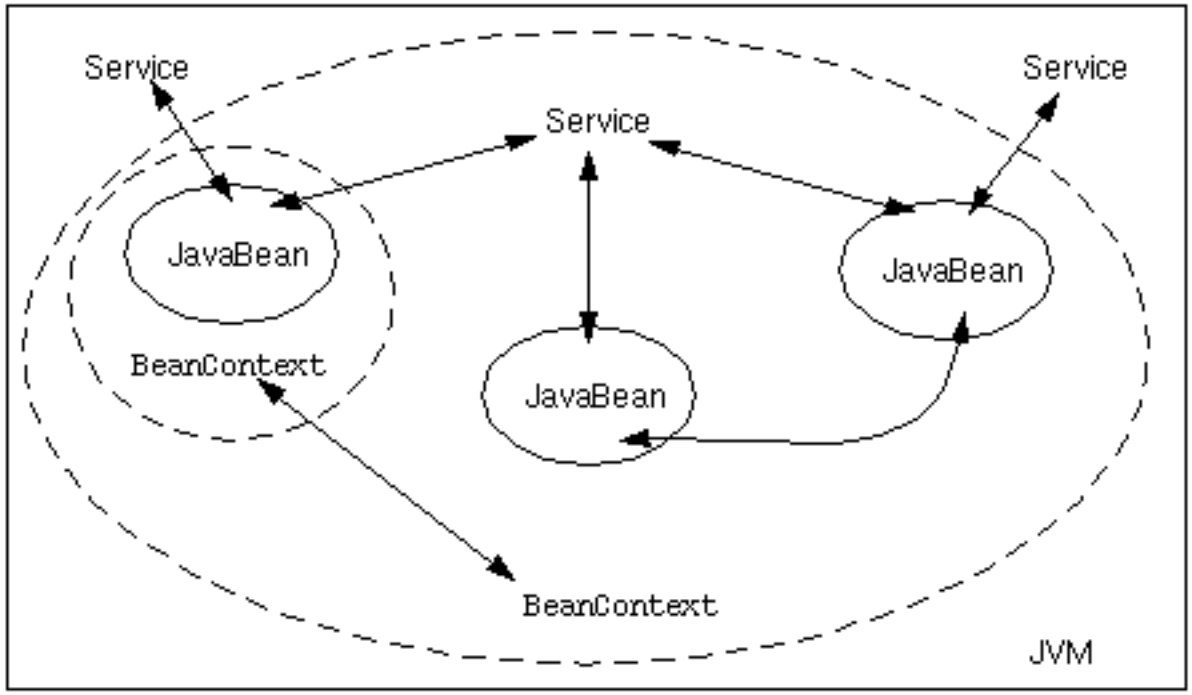
Miscellaneous

- `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

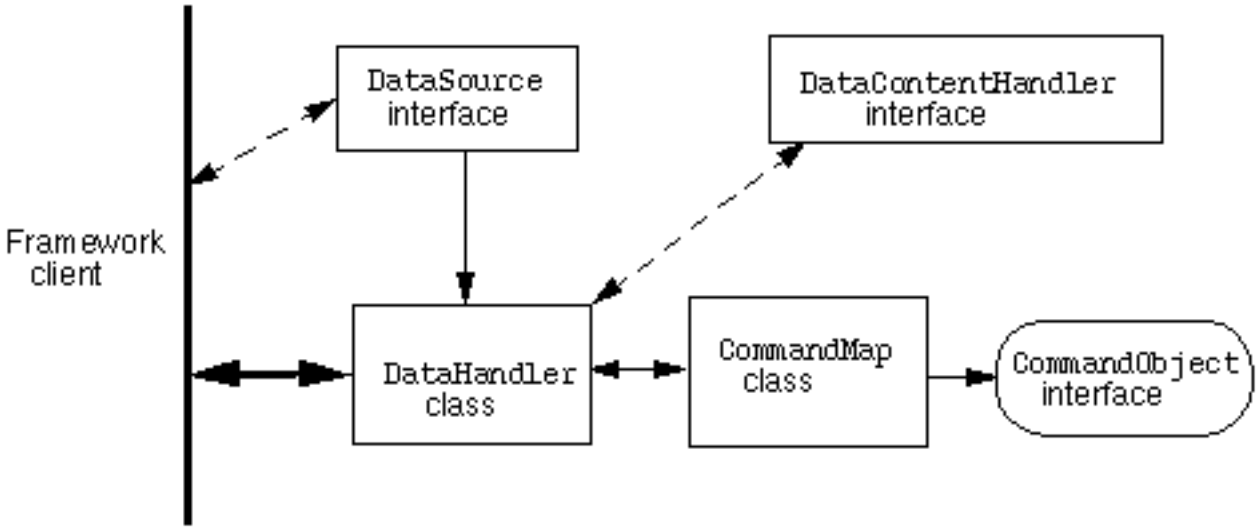
`BeanContext` Support Classes

- `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`.