JavaBeans

- Introduction to component-based software architecture
- Features of a Component
- Components in a Framework
- Overview of JavaBeans
- Design issues of JavaBeans
- Java BDK1.1
- Summary

Component-based Software Architecture

- The next wave of software revolution
  - Transition from Objects to Components
    - OLE2.0, OpenDoc Parts, Java Beans, ActiveX
  - Transition from Client/Server to DOC
    - CORBA, COM/DCOM, InfoBus, ...
- Combination of both creates a true Plug-and-Play architecture
  - Modular, Scalable, Flexible, Manageable
Features of a Component

Basic features:
- capable of describing itself
- capable to interact with the framework/other components via events
- be able to be plugged into a framework

Configurable by a editor/builder tool
- HyperCard, JBuilder, VB, VC++ resource editor

Deployable as a unit or in a container package (jar, dll, etc.)

Components in a Framework

Components are “plugged” into a particular framework via standard interfaces
Overview of JavaBeans

- What is JavaBeans?
  - A concept for writing component software
- A Java bean is a class (can be visible or invisible) with the following features using standard coding conventions:
  - Properties, Methods and Events
- Additional classes/interfaces to support
  - Reflection, Introspection, Customization and Persistence

The Object Model of JavaBeans

```
java.lang.object
  |-- java.util.EventObject
    |  |-- MyEvent
    |     |-- MySourceBean
    |     |  |-- java.util.Vector
    |     |     |-- MySourceBeanBeanInfo
    |     |     |  |-- MySourceBeanCustomizer
    |     |     |  |-- MySourceBeanBeanInfo
    |     |     |-- MyListenerBean
    |     |     |  |-- MyListenerBeanBeanInfo
    |     |     |  |-- MyListenerBeanCustomizer
    |     |  |-- java.util.EventListener
    |     |     |-- MyEventListener
    |     |     |-- MySourceBeanBeanInfo
    |     |     |  |-- MySourceBeanCustomizer
    |     |     |  |-- MySourceBeanBeanInfo
    |     |     |-- MyListenerBean
    |     |     |  |-- MyListenerBeanBeanInfo
    |     |     |  |-- MyListenerBeanCustomizer
```
Design issues of JavaBeans

- Bean Properties (What do I have)
- Bean Methods (What I am capable of)
- Event Handling (What message I can send)
- Persistency (same as object persistency)
- Support classes:
  - Explicit bean info via `BeanInfo` interface
  - Customization via `Customizer` interface
  - Editor support via `PropertySupportEditor`

Bean Properties

A property represents an attributes of a Bean object. It is implemented as a data member with accessors using standard coding convention.

Four types of properties are defined:
- Simple
- Indexed
- Bound
- Constrained
Simple and Indexed Properties

- Simple: data member of a non-array type
  - int size;
  - int getSize() { return size; }
  - void setSize(int s) { size = s; }

- Indexed: data member of an array type
  - int[] sizes;
  - int[] getSizes(); void setSizes(int s[]);
  - int getSizes(int index);
  - void setSizes(int val, int index);

Bound and Constrained Properties

- Bound: the setXXX() method will fire a PropertyChangeEvent to registered PropertyChangeListeners of this bean to inform them (listeners) about the change.

- Constrained: the setXXX() method will ask a list of VetoableChangeListeners about the proposed changes. If no objection, i.e. no PropertyVetoException is thrown, the corresponding value will be changed.
Event handling in JavaBeans

In addition to PropertyChangeEvent, a bean component is able to generate its own event(s) using JDK1.1 event model.

Three parts need to be dealt with in a Java bean component:
- Event Object(s), Event Listener(s), Event manipulation function(s)

A bean component needs to maintain a list of event listener(s)

Creating User-defined Events

```java
public class TaxEvent extends java.util.EventObject {
    private String ssn; double taxAmount;
    public TaxEvent
        (Object source, double amount, String s) {
            super (source); // specify the event source
            taxAmount = amount;  ssn = s;
        }
    public double getTaxAmount () { return taxAmount; } 
    public ssn getSsn() { return ssn; } 
}
```
Event Listener

```java
public interface TaxListener
extends java.util.EventListener
{
    public void processTaxFiling (TaxEvent e);
}
```

Maintaining a listener list

```java
public class TaxPayer {
    String name, ssn, taxYear; ......
    Vector taxAgencies;
    public TaxPayer(String n, String s, String y) {
        ssn = s; name = n; taxYear = y; ....
        taxReceiver = new Vector();
    }
    public synchronized addTaxListener(TaxListener l) {
        taxAgencies.add(l);
    }
    public synchronized removeTaxListener(TaxListener l) {
        taxAgencies.remove(l);
    }
}....
```
Firing an Event

```java
protected void filingTaxes () {
    Vector l;
    TaxEvent te = new TaxEvent (this, amount, ssn);
    // Copy listener vector so it won't change while firing
    synchronized (this) {
        l = (Vector)hireListeners.clone();
    }
    for (int i=0;i<l.size();i++) {
        TaxListener tl = (TaxListener)l.elementAt (i);
        tl.processTaxFiling(te);
    }...
}
```

Methods of a Bean

- `getXXX` methods and `setXXX` of properties
- Public methods that deal with bean events
- Public methods for receiving events
- Public or non-public functions with non-arguments to support bean builder/editors
- One should avoid using other types of public methods with arguments. It would complicate the process of inspecting the bean by a bean builder
Classes and interfaces to support bean builders/editors

- The *Customizer* interface: to allow customization of the UI of a bean editing panel inside a bean editor
- The *PropertySupportEditor* class: to provide customized values of a property.
- The *BeanInfo* interface and the *SimpleBeanInfo* class: to provide explicit information about a bean (properties, events, methods) to a bean editor

Packaging and deploy Java beans

- Beans are packaged in Java Archive (jar)
  - jar file: in zip format, but with a manifest file
  - jar command: part of the JDK tools, similar syntax as UNIX tar
- Manifest file format for Java beans:
  - you need to create a entry for each class file
    *Name*: `package_dir/classname.class`
    
    *Java-Bean*: True
  - `jar cfm jarfile.jar manifest package_dir/*.class`
BDK (Bean Development Kit) 1.1

- Sun’s bean testing and integration tool
- Contains three parts:
  - ToolBox: displays a list of beans
  - BeanBox: GUI for rendering visible beans
  - Property editor: for editing bean’s properties
- Beans are stored in a jar file
- Possible to add your own bean to the bean box by creating a jar file
- More info at http://java.sun.com/beans

Technologies based on JavaBeans

- Glasgow
  - Extensible Runtime Containment and Services Protocol (protocol to support embedded beans)
  - Drag and Drop Subsystem for the Java Foundation Classes (already supported by JDK1.2)
  - JavaBeans Activation Framework (link type info with services)
- InfoBus - A framework to enable data exchange between enterprise beans
Summary

- JavaBeans provides a methodology to develop reusable component software.
- A bean component has to be developed with properties, events, methods and supporting classes using standard coding convention.
- Design of components needs to be conducted in conjunction with the development of application frameworks and standards of software industry.