

CmpE 473

Internet Programming

Pınar Yolum
pyolum@cmpe.boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Chapter 5

Java Beans

What are beans? (1)

- Reusable Java code
- Easiest to reuse is static objects;
 - Ex: Text fields, buttons
 - Available as third-party products
 - Limited interaction with environment
 - The functionality does not change
- Can be composed into complex beans
- Drag-and-drop using application construction tools

Spring 2005— Pinar Yolum

3

What are beans? (2)

- Beans have properties
 - Related to appearance
 - Customizable
- Communicate to other beans with events
 - Listener bean must register with the source bean to receive events
- Well-defined, standard design interface
 - Query properties
 - Query events they generate or respond to

Spring 2005— Pinar Yolum

4

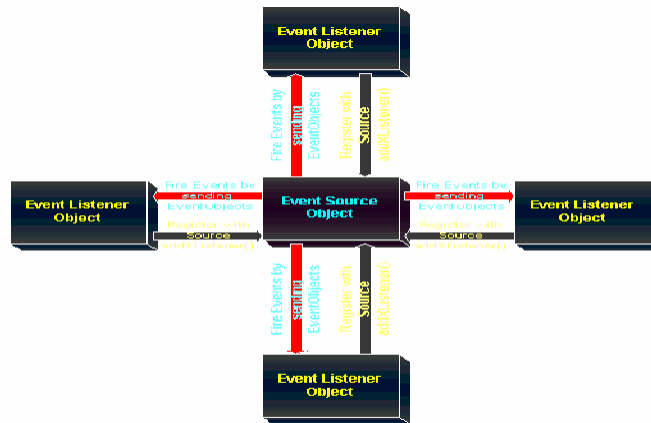
What are beans? (3)

- Enable easy manipulation by tools
- Introspection: Methods are specified in a specific way so that bean tools can look into them
- Persistence: Bean states can be saved and restored (they implement `java.io.Serializable`)

Events

- Beans communicate through events
 - `EventObject`
 - `EventListener` - (the sink)
 - An Event Source (the Bean)
- `EventObjects` represent events
 - Bankaccount changed
 - Mouse click
- `EventListener` expects to be notified when there is a change
- `EventSource` announces events
 - Classes register to hear about the event

Events



- <http://my.execpc.com/~gopalan/java/dragger.html>

Spring 2005— Pinar Yolum

7

Events

1. Event occurs
2. Event source broadcasts the event to all event listeners
3. Event object is passed to the listeners as argument
4. Event listeners act as they like (e.g., change the form, show a messages)

Spring 2005— Pinar Yolum

8

Event Object

- `java.util.EventObject`

```
public class java.util.EventObject extends Object implements java.io.Serializable {  
    public java.util.EventObject (Object source);  
    public Object getSource();  
    public String toString();  
}
```
- New event extends `EventObject` (Ex: `javax.print.event.PrintJobEvent`)

```
public class HireEvent extends EventObject {  
    private long hireDate;  
    public HireEvent (Object source) {  
        super (source);  
        hireDate = System.currentTimeMillis();  
    }  
    public HireEvent (Object source, long hired) {  
        super (source);  
        hireDate = hired;  
    }  
    public long getHireDate () {  
        return hireDate;  
    }  
}
```

Spring 2005— Pinar Yolum

9

Event Listener

- Event objects are received at the listeners
- Name of the listener: *eventType*Listener that extends `EventListener`
- Eventlistener is just a marker (no methods)

```
public interface HireListener extends  
    java.util.EventListener {  
    public abstract void hired (HireEvent e);  
}
```

Spring 2005— Pinar Yolum

10

Event Source

- Defined by the user
- Defines when the event will happen
- Registration process: (Add/remove listeners)
private Vector hireListeners = new Vector();
public synchronized void addHireListener (HireListener l) {
 hireListeners.addElement (l);
}
public synchronized void removeHireListener (HireListener l) {
 hireListeners.removeElement (l);
}
}
- Throw `java.util.TooManyListenersException` if you want to serve one listener

Spring 2005— Pinar Yolum

11

Notifying

```
protected synchronized void notifyHired () {  
    Vector l;  
    // Create Event  
    HireEvent h = new HireEvent (this);  
    l = (Vector)hireListeners.clone();  
    for (int i=0;i<l.size();i++) {  
        HireListener hl = (HireListener)l.elementAt (i);  
        hl.hired(h);  
    }  
}
```

Spring 2005— Pinar Yolum

12

Properties

- Simple
- Indexed
- Bound
- Constrained

Simple Property

- Define a set of set/get methods
- `getX/setX`: X becomes the property name

```
float salary;
public void setSalary (float newSalary) {
    salary = newSalary;
}
public float getSalary () {
    return salary;
}
```

- Instead of `getX`, `isX` used for booleans

```
public boolean isTrained () {
    return trained;
}
```

Indexed Property

- Single property holds an array of values
- Pattern follows these:

```
public void setPropertyName (PropertyType[] list)
public void setPropertyName ( PropertyType element, int
position)
public PropertyType[] getPropertyName ()
public PropertyType getPropertyName (int position)
```

Indexed Property

- Example: (using java.awt.List)

```
public class ListBean extends List {
    public String[] getItem () {
        return getItems ();
    }
    public synchronized void setItem (String item[]) {
        removeAll();
        for (int i=0;i<item.length;i++) addItem (item[i]);
    }
    public void setItem (String item, int position) {
        replaceItem (item, position)
    }
}
```

- The String getItem (int position) routine already exists for List.

Bound Property (1)

- **Bound Properties**
 - Set some properties as bound properties
 - Identify listener beans
 - When bound properties change, listener beans are notified
 - The listener can perform an action on receiving this event

Bound Property (2)

- **Create list of listeners**

```
private PropertyChangeSupport changes = new PropertyChangeSupport (this);
```
- **Add/Remove from the list**

```
public void addPropertyChangeListener ( PropertyChangeListener p) {  
    changes.addPropertyChangeListener (p);  
}  
public void removePropertyChangeListener ( PropertyChangeListener p) {  
    changes.removePropertyChangeListener (p);  
}
```

Bound Property (3)

- Fire changes when needed

```
public void setSalary (float salary) {  
    Float oldSalary = new Float (this.salary);  
    this.salary = salary;  
    changes.firePropertyChange ( "salary", oldSalary, new Float  
    (this.salary));  
}
```

- Receive the event on the waiting side

```
public void propertyChange(PropertyChangeEvent e);
```

- Check at the receiving side if you got the event for a property you were expecting (property changes are sent at the level of the bean)

Constrained Property (1)

- Constrained Properties

- Listener or the source bean can veto a change by throwing `PropertyVetoException`
- Add listeners for the property
- The listeners have to implement `VetoableChangeListener` interface
- Beans keep track of the `VetoableChangeListeners`
- A source bean defines a constrained property
- The listener can perform an action on receiving this event

Constrained Property (2)

- Add to the previous example

```
private VetoableChangeSupport vetoes = new VetoableChangeSupport (this);
public void addVetoableChangeListener (VetoableChangeListener v) {
    vetoes.addVetoableChangeListener (v);
}
public void removeVetoableChangeListener (VetoableChangeListener v) {
    vetoes.removeVetoableChangeListener (v);
}
```

- Change in the example

```
public void setSalary (float salary) throws PropertyVetoException {
    Float oldSalary = new Float (this.salary);
    vetoes.fireVetoableChange ( "salary", oldSalary, new Float (salary));
    this.salary = salary;
    changes.firePropertyChange ( "salary", oldSalary, new Float (this.salary));
}
```

- Again the receiving side must have

```
public void vetoableChange(PropertyChangeEvent e) throws PropertyVetoException;
```

Method

- Public methods called by all
- Event source calls appropriate method to notify
- Support methods to allow builder applications to connect to
 - No arguments
 - An argument of the event you listen to

Introspector

- Discover a bean's configurable properties
- Used manually or through an IDE
- Bean-info classes implement bean-info interface
 - Publish list of configurable properties
 - List of accessor methods
 - If bean-info class is absent, use Reflection
- Bean-info classes are appended with bean-info (Ex: TextFieldBeanInfo)

Introspector

- `TextField tf = new TextField();`
`BeanInfo bi = Introspector.getBeanInfo(tf.getClass());`
- SimpleBeanInfo class provides empty methods to be overwritten

```
package hire;
import java.beans.*;
public class HireBeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        try {
            PropertyDescriptor pd1 = new PropertyDescriptor("salary",
Hire.class);
            pd1.setBound(true);
            return new PropertyDescriptor[] {pd1};
        } catch (Exception e) { return null; }
    }
}
```