Administrivia

- Project 6 spec has been updated
  - file sharing registration now takes a URL object, not a File
  - don’t need to run a web server, can put shared files on WAM web site, or wherever
    - provided listener implementation just prints URL
  - sample startup script posted

Last time

- JavaBeans
  - Software components in Java
  - Event model
    - same as for AWT and Swing GUI libraries
    - listeners register with a bean to be notified of events
  - Properties – get and set
    - bound – other beans notified of a property change
    - constrained – other beans can veto a property change
  - Introspection
    - use Java reflection to find out about a bean’s properties, events, other methods
  - Persistence
    - all beans can be serialized and deserialized

Simple Bean example

```java
import java.awt.*;
import java.io.Serializable;
public class SimpleBean extends Canvas implements Serializable {
    private Color color = Color.green;

    public Color getColor() {
        return color;
    }

    public void setColor(Color newColor) {
        color = newColor;
        repaint();
    }
}
```

Properties

- If a component supports functions:
  - public void setMyValue(int v)
  - public int getMyValue()
- It has a MyValue property of type int
- For boolean types, getter function can be named is<Prop>()
- Can have read-only, read/write or write-only properties
  - don’t have to define both getter and setter method

Example, with Simple Property

```java
import java.awt.*; import java.io.Serializable;
public class SimpleBean extends Canvas implements Serializable {
    private Color color = Color.green;
    // property getter method
    public Color getColor() {
        return color;
    }
    // property setter method. Sets new SimpleBean
    // color and repaints.
    public void setColor(Color newColor) {
        color = newColor;
        repaint();
    }
}
```
Simple Property, cont.

```java
public void paint(Graphics g) {
    g.setColor(color);
    g.fillRect(20, 5, 20, 30);
}

// Constructor sets inherited properties
public SimpleBean() {
    setSize(60, 40);
    setBackground(Color.red);
}
```

Java Bean Event Patterns

- A Bean Event must extend
  ```java
  class java.util.EventObject {
      public EventObject(Object src);
      public Object getSource();
  }
  ```
- Name should end in Event
  e.g., tempChangeEvent

Event Listeners

- must implement java.util.EventListener
  just a marker interface
- have event-Listener methods
  ```java
  void eventName(<EventType> e);
  ```
- interface TempChangeListener
  ```java
  void tempChanged(TempChangedEvent e);
  ```

Event sources

- Event sources fire events
- Have methods to attach/detach Listeners
  ```java
  public void add<ListenerType>(ListenerType ls);
  public void remove<ListenerType>(ListenerType ls);
  ```

Event Adapters

- Easy to construct event adapters
  For example, an adapter that receives
temperatureChanged events, and generates
temperatureIncreased and
temperatureDecreasedEvents

Bound properties

- Can set things up so that changes to Bean property
  are indicated by an event
  after the change occurs
- these events are a subtype of java.beans.PropertyChangeEvent
- Listeners implement PropertyChangeListener and the
  propertyChange method is invoked when the event is
  fired
- One Listener for all change events on the bean
  may optionally support listeners for specific properties
Bound Property support

- Convenience class `PropertyChangeSupport`
  - implements methods to add/remove `PropertyChangeListener` objects and fire `PropertyChangeEvent` objects at listeners when the bound property changes

Implementing a Bound Property

```java
import java.beans.*
public class Bound … {
    // instantiate PropertyChangeSupport object
    private PropertyChangeSupport changes =
        new PropertyChangeSupport(this);
    // methods to implement property change listener list
    public void addPropertyChangeListener(  
        PropertyChangeListener l) {
        changes.addPropertyChangeListener(l);}
    public void removePropertyChangeListener( 
        PropertyChangeListener l) {
        changes.removePropertyChangeListener(l);}
}
```

Bound Properties, cont.

```java
// modify property setter method to fire PropertyChangeEvent
public void setLabel(String newLabel) {
    String oldLabel = label;
    label = newLabel;
    sizeToFit();
    changes.firePropertyChange("label", oldLabel, newLabel);}
```

Creating a Listener

```java
// implement the PropertyChangeListener interface
public class MyClass implements java.beans.PropertyChangeListener {
    void propertyChange(PropertyChangeEvent evt) {
        // handle a property change event
        // e.g., call a setter method in the listener class
    }
}

// and register the listener with the source Bean
button.addPropertyChangeListener(aButtonListener);
```

Constrained Properties

- Source Bean contains one or more `constrained` properties
  - should also usually be bound properties
- Listeners can veto property changes
  - before the actual property change occurs
  - implement `VetoableChangeListener` interface
  - Listener throws `PropertyVetoException`
  - set<Property> method throws …

Constrained Properties

```java
import java.beans.*
public class Constrained … {
    // instantiate VetoableChangeSupport object
    private VetoableChangeSupport vetos =
        new VetoableChangeSupport(this);
    // methods to implement property change listener list
    public void addVetoableChangeListener(  
        VetoableChangeListener l) {
        vetos.addVetoableChangeListener(l);}
    public void removeVetoableChangeListener(   
        VetoableChangeListener l) {
        vetos.removeVetoableChangeListener(l);}
```
Constrained Properties, cont.
// modify property setter method to fire PropertyChangeEvent
// including adding throws clause
public void setPriceInCents(int newPriceInCents )
throws PropertyVetoException {
    int oldPriceInCents = ourPriceInCents;
    // First tell the vetoers about the change.
    // If anyone objects, don't catch the exception
    // but just let it pass on to the caller
    vetos.fireVetoableChange("priceInCents",
        new Integer(oldPriceInCents ),
        new Integer(newPriceInCents ));
    // No one vetoed, so go ahead and make the change.
    ourPriceInCents = newPriceInCents ;
    changes.firePropertyChange("priceInCents",
        new Integer(oldPriceInCents ),
        new Integer(newPriceInCents ));
}

Constrained Properties, cont.
// this builds a PropertyChangeEvent object, and calls
// vetoableChange(PropertyChangeEvent pce) on each registered
// listener
public void fireVetoableChange(String propertyName,
    Object oldValue,
    Object newValue)
throws PropertyVetoException

Creating a Listener
• Same as for PropertyChangeListener
  – listener Bean implements
    VetoableChangeListener interface
  – with method
    vetoableChange(PropertyEvent evt)
    throws PropertyVetoException;
  • called by source Bean on each registered listener
    object, and exercises veto power by throwing the
    PropertyVetoException

Serialization and Persistence
• Can manipulate Java Beans in a builder tool
• Doesn’t help if can’t distribute the beans
• Serialize the beans
  – Bean must implement java.io.Serializable or
    java.io.Externalizable (to get complete control
    over the serialization)
• Application loads beans from Serialized form

Default Serialization
• Beans that implement Serializable must have a
  no-argument constructor
  – to call when reconstituting the object
• Don’t need to implement Serializable if already
  implemented in a superclass
  – unless need to change the way it works
• All fields except static and transient ones are
  serialized
  – default serialization ignores those fields
  – transient also can mark an entire class as not
    serializable

Selective Serialization
• To override default serialization, implement
  ReadObject() and/or WriteObject()
  – to exercise complete control over what gets serialized
  – to serialize objects default serialization can’t handle
  – to add data to serialization stream that is not a field in
    the object
  – if override WriteObject(), override ReadObject() too
  – private void writeObject(java.io.ObjectOutputStream out)
    throws IOException;
  – private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;

CMSC 433, Spring 2001 - Alan Sussman 19
CMCS 433, Spring 2001 - Alan Sussman 20
CMCS 433, Spring 2001 - Alan Sussman 21
CMCS 433, Spring 2001 - Alan Sussman 22
CMCS 433, Spring 2001 - Alan Sussman 23
CMCS 433, Spring 2001 - Alan Sussman 24
Example – Molecule Demo Bean

```java
private void writeObject(java.io.ObjectOutputStream s) throws java.io.IOException {
    s.writeInt(ourVersion); // a static field
    s.writeObject(moleculeName); // a class field
}
private void readObject(java.io.ObjectInputStream s) throws java.lang.ClassNotFoundException,
java.io.IOException {
    // Compensate for missing constructor
    reset();
    if (s.readInt() != ourVersion) {
        throw new IOException("Molecule.readObject:
        version mismatch");
    }
    moleculeName = (String) s.readObject();
}
```

Default Read/Write Object

```java
private void writeObject(java.io.ObjectOutputStream s) throws java.io.IOException {
    //First write out defaults
    s.defaultWriteObject();
    //... }
private void readObject(java.io.ObjectInputStream s) throws java.lang.ClassNotFoundException,
java.io.IOException {
    //First read in defaults
    s.defaultReadObject();
    //... }
```

Externalizable interface

- To get complete control over Bean’s serialization
  - e.g., for writing/reading a specific file format
  - implement `readExternal()` and `writeExternal()`
  - these classes also require a no-argument constructor

CMSC 433, Spring 2001 - Alan Sussman