



Course Name:
Advanced Java



Lecture 23

Topics to be covered

- Using Beans to Build an Application
- Naming Patterns for Bean Components
- Events Bean Property Types



Using Beans to Build an Application

- Builder environments aim to reduce the amount of drudgery that is involved in wiring together components into an application.
- Each builder environment uses its own strategies to ease the programmer's life. The NetBeans is better integrated development environment because it is a fairly typical programming environment & it is freely available also.



- **Packaging Beans in JAR Files –**

- To make any bean usable in a builder tool, package into a JAR file all class files that are used by the bean code.
- A JAR file for a bean needs a manifest file that specifies which class files in the archive are beans & should be included in the ToolBox.
- If your bean contains multiple class files, just mention in the manifest those class files that are beans & that you want to have displayed in the toolBox.

To make the JAR file, follow these steps :

- 1) Edit the manifest file.
- 2) Gather all needed class files in a directory.
- 3) Run the jar tool as follows:

```
jar cvfm JarFile ManifestFile ClassFiles
```

Ex:

```
jar cvfm ImageViewerBean.jar  
ImageViewerbean.mf  
com/horstmann/corejava/*.class
```

You can also add other items such as GIF files for icons, to the JAR file.



Builder Environments have a mechanism for adding new beans, typically by loading JAR files.

Here is what you do to import beans into NetBeans.

Compile the ImageViewerBean & FileNameBean classes & package them into JAR files.

Then start NetBeans & follow these steps :

- 1) Select Tools --> Palette Manager from the menu.
- 2) Click the Add from JAR button.

In the file dialog box, move to the ImageViewerBean directory & select ImageViewerBean.jar

Now a dialog box pops up that lists all the beans that were founding the JAR file. Select ImageViewerBean.

Finally, you asked into which palette you want to place the beans. Select Beans.

Have a look at the Beans palette. It now contains an icon representing the new bean.

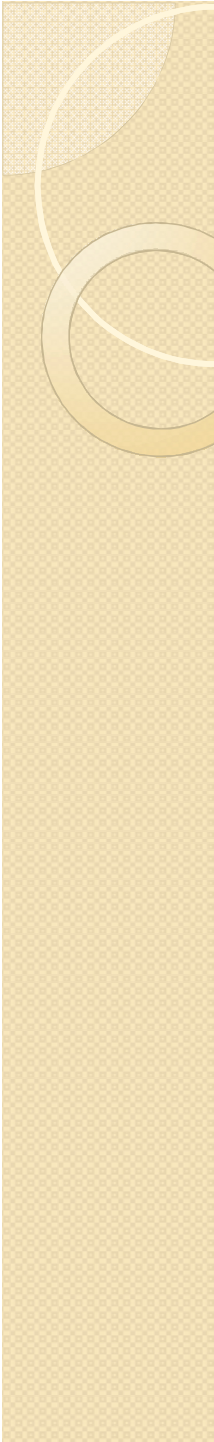


Naming Pattern for Bean Properties and Events

- There is no cosmic beans class that you extend to build your beans. Visual beans directly or indirectly extend the Component class.
- But non visual beans don't have to extend any particular superclass.
- There are two alternatives mechanisms:
 - If the bean writer uses standard naming patterns for properties & events, then the builder tool can use the reflection mechanism to understand what properties & events the bean is supposed to expose.
 - The bean writer can supply a bean information class that tells the builder tool about the properties & events of the bean.

Naming pattern for properties

- The get method is named `get<PropertyName>()`,
which takes no parameters and returns an object of the type identical to the property type.
- For a property of boolean type, the get method should be named `is<PropertyName>()`,
which returns a boolean value.
- The set method should be named `set<PropertyName>(newValue)`,
which takes a single parameter identical to the property type and returns void.

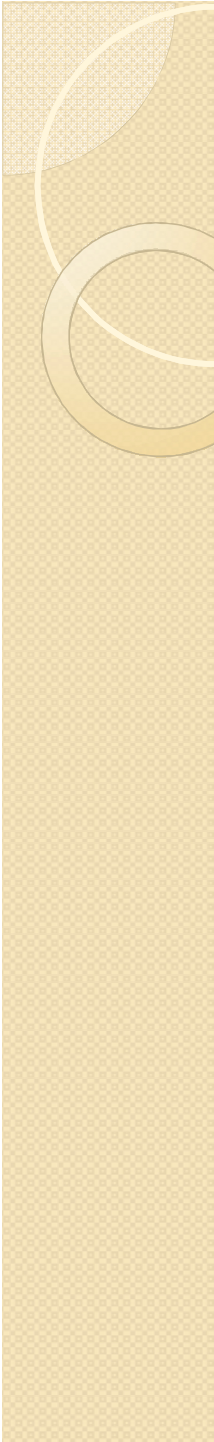
- 
- If u have a get method but not an associated set method, u define a read-only property.
 - If u have a set method without an associated get method then it defines a write only method.

 - An animation might have a property running with two methods:
 - `public boolean isRunning()`
 - `public void setRunning(boolean b)`
 - The `setRunning` method would start & stop the animation.
 - The `isRunning` method would report its current status.



Naming Pattern for Events

- A bean builder environment will infer that your bean generates events when you supply methods to add & remove eventlisteners.
- All event class names must end with in Event, & the classes must extend the EventObject class.

- 
- Suppose your bean generates events of type `EventNameEvent`. The listener interface must be called `EventNameListener` & the methods to add & remove a listener must be called.
 - `public void`
 `addEventNameListener(EventNameListener e)`
 - `public void`
 `removeEventNameListener(EventNameListener`
 `e)`



Properties

- Discrete, named attributes that determine the appearance and behavior and state of a component
- Accessible programmatically through accessor methods
- Accessible visually through property sheets
- Exposed as object fields in a scripting environment

Bean Property Types

- A bean has a lot of different kinds of properties that it should expose in builder tool for a user to set at design time or get at run time.
- It also triggers both standard & custom events.
- **The Java bean specification allows four types of Properties:**
 - Simple Properties
 - Indexed Properties
 - Bound Properties
 - Constrained Properties

Simple Properties

- Represent a single value
- The accessor methods should follow standard naming conventions

```
public <PropertyType> get<PropertyName>();  
public void set<PropertyName>(<PropertyType> value);
```

Example:

```
public String getHostName();  
public void setHostName( String hostName );
```

Boolean Properties

- They are simple properties
- The getter methods follow an optional design pattern

```
public boolean is<PropertyName>();
```

Example:

```
public boolean isConnected();
```

Indexed Properties

- Represent an array of values

```
public <PropertyElement> get<PropertyName>(int index);  
public void set<PropertyName>(int index,<PropertyElement> value);  
public <PropertyElement>[] get<PropertyName>();  
public void set<PropertyName>(<PropertyElement>[] values);
```

Example:

```
public Color setPalette(int index);  
public void setPalette(int index, Color value);  
public Color[] getPalette();  
public void setPalette(Color[] values);
```


Bound Properties

- Registered listeners object are notified when the value of the property changes
- Listeners must implement the *java.beans.PropertyChangeListener* interface

```
propertyChange(PropertyChangeEvent event);
```

Constrained Properties

- Allow registered listeners to validate a proposed change
- Listeners must implement the *java.beans.VetoableChangeListener* interface

```
vetoableChange( PropertyChangeEvent event )  
                throws PropertyVetoException;
```

Constrained Properties - Example

```
public void setHostName( String newHostName )
    throws java.beans.PropertyVetoException
{
    String oldHostName = this.hostName;

    // First tell the vetoers about the change.  If anyone objects, we
    // don't catch the exception but just let it pass on to our caller.
    vetoableChangeSupport.fireVetoableChange( "hostName",
        oldHostName, newHostName );

    // change accepted; update state
    this.hostName = newHostName;

    // notify property change listeners
    propertyChangeSupport.firePropertyChange("hostName",
        oldHostName, newHostName );
}
```