Introduction to Java Beans

Author Name: G. Sreenivasulu
Department: CSE

Web Technologies

Session Objectives

At the end of this session, the learner will be able to:

- Define a Reusable Software Component
- The Technology alternatives for reusable software components
- Define a Java Bean
- Features of Java Bean
- Properties of a Java Bean

Teaching Learning Material

- Story Telling
- White Board and Markers
- Presentation Slides
- LCD Projector

Session Plan

Time (in min)	Content	Learning Aid and Methodology	Faculty Approach	Typical Student Activity	Learning Outcomes (Blooms + Gardeners)
15	Define Reusable Software Component	Story Telling Model Demonstration	Facilitates Explains	Listens, Participates Relates Absorbs	Understand Intra Personal Inter personal Logical
10	The Technology alternatives for reusable software components	Chalk & Talk	Facilitates Explains	Listens, Participates	Understand Logical Intra Personal
5	Define a Java Bean	PPT Chalk & Talk	Facilitates Explains	Listens	Understand Apply Intrapersonal
5	Features of Java bean	Questioning Chalk & Talk	Facilitates Explains	Listens, Participates Relates Absorbs	Understand Inter-personal Intra Personal Analyze
15	Properties of a Java Bean	Chalk & Talk	Explains	Listens, Relates	Understand Interpersonal Intrapersonal Debate

Session Inputs

Define Reusable Software Component:

Notes

Software component reuse is the software engineering practice of creating new software applications from existing components, rather than designing and building them from scratch. In order to achieve greater productivity and easier maintenance

Suggested Activity: Story-Telling / Role Play

To simplify the concept of reusable software component Faculty uses a small story of to explain about reusability of components in various areas.

Also there is role play in which a person, who has just started construction of a new house is taking suggestions from his friend. Two learners will volunteer this story

Learner 1: You know I am constructing a new house.

Learner 2: Yes, tell me do you need any suggestions.

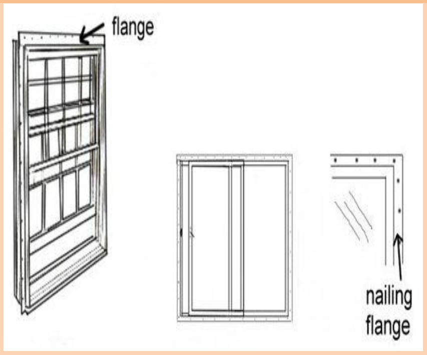
Learner 1: I am thinking of finishing to complete the works fast

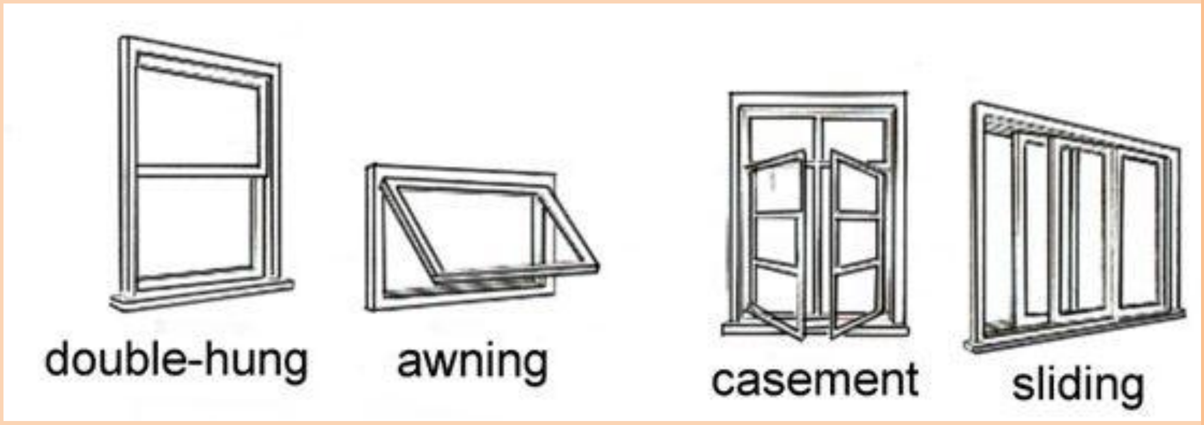
Learner 2: Now a days lot of prefabricated construction material is Available. You have ready made doors, windows, roofs walls What not. Even ready mix concrete.

Learner 1: oh, it is great.

Learner 2: But there is a problem with these prefabricated

The role play and the above story suggests, that apart from many benefits we also have problem of not suiting to changed specifications.





Announcement

Component or service that can be used as a part of business logic of other components

The Technology alternatives for reusable software components:

Notes

It is necessary to have reusable software components which can be used in various technologies in which you are working.

Suggested Activity: Chalk & Talk

Faculty explains about the need for alternatives and availability of alternative technologies.

Announcement

There are two technology alternatives first is Microsoft's ActiveX and the second one is Java Beans.

Define a Java Bean:

After Knowing what is a reusable software component and after knowing two technology alternatives in reusable components, we now will see how to create a reusable software component in Java i.e. Java bean.

Suggested Activity: PPT , Chalk & Talk

Faculty will present the program code for creating a java bean. Which has the following steps?

Steps to create a Java Bean

1. Write a Java bean Class, save it under a file with . java Extension

example: SimpleBean.java

```
   import java.awt.*;
   import java.io.Serializable;

   public class SimpleBean extends Canvas implements Serializable {
      //Constructor sets inherited properties
      public SimpleBean() {
         setSize(60,40);
         setBackground(Color.red);
       }

private Color beanColor = Color.green;

      public void setBeanColor(Color newColor) {
         Color oldColor = beanColor;
         beanColor = newColor;
         repaint();

         // This relates to bound property support (see below).
         changes.firePropertyChange("beanColor", oldColor, newColor);
```

```
        }

        public Color getBeanColor() {
           return beanColor;
        }

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

private PropertyChangeSupport changes = new
PropertyChangeSupport(this);

        public void addPropertyChangeListener(PropertyChangeListener l) {
           changes.addPropertyChangeListener(l);
        }

        public void removePropertyChangeListener(PropertyChangeListener l)
{
           changes.removePropertyChangeListener(l);
        }
public void propertyChange(PropertyChangeEvent evt) {
           String propertyName = evt.getPropertyName();
           System.out.println("Received property change event " +
propertyName);
        }

    }
```

2. Compile the .java file (in our example SimpleBean.java)

```
        > javac SimpleBean.java
```

3. Create Manifest File manifest.tmp as shown below

<u>manifest.tmp</u>

```
Name: SimpleBean.class
Java-Bean: True
```

4. Java Bean must be compiled and packaged into a JAR file, jar file can be created as given below

```
> jar cfmv SimpleBean.jar manifest.tmp SimpleBean.class
```

A Java Bean is a reusable software component that can be manipulated visually in an application builder tool. The idea is that one can start with a collection of such components, and quickly wire them together to form complex programs without actually writing any new code.

Features of Java bean:

We have seen how to create a Java Bean with, now we must try to know various features of a Java Bean.

Suggested Activity: Questioning , Chalk & Talk.

The learners will be asked few questions given below and the faculty will write the answers. Faculty will write the list of Java bean features with answers from learners.

Learners will be asked following questions.

1. What represents the state of an Object (Class)?
2. What represent the behavior of an object?
3. What is persistence?
4. Have you noticed any special convention in method naming?

Expected answers:

1. State of an object is given by the data it contains at any point of time.
2. Behavior of an object is the set of actions that it can perform.

3. Persistence enables beans to save and restore their state
4. The Java Bean spec defines the entities described by get and set accessors as "properties". By using the simple get/set naming convention, any tool that supports JavaBeans can determine which properties are available for a bean just by looking at the names of the provided methods

Announcement

Java Bean Features

- JavaBeans support properties, allowing an application to read and modify their values.
- JavaBeans support events, allowing vendors to create their own unique events.
- JavaBeans support the BeanInfo interface, allowing vendors to specify exactly which properties and methods are available, and icons for beans which can be displayed on a toolbar.
- JavaBeans are highly configurable, and the state of a bean can be saved and restored through 'serialization'

Properties of a Java Bean:

Notes

After knowing the definition of Java Bean Properties, it is important to know various properties of Java Beans. Also a brief explanation about various types of properties.

Suggested Activity: Chalk & Talk

Faculty explains various types of properties. Learners will listen and relate them to the mechanism used to define and access properties in Java.

Simple Properties:

```
public class MyBean {

 public MyBean() {
    }
    private String title;
    public String getTitle() {
        return this.title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
}
```

Indexed Properties

```
private String[] lines;
    public String getLines(int index) {
        return this.lines[index];
    }
public String[] getLines() {
        return this.lines;
    }
 public void setLines(int index, String lines) {
        this.lines[index] = lines;
    }
public void setLines(String[] lines) {
        this.lines = lines;
    }
   }
```

Bound Properties:

```
import java.awt.Graphics;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.io.Serializable;
import javax.swing.JComponent;


public class MyBean
```

```
      extends JComponent
      implements Serializable
{
   private String title;
   private String[] lines = new String[10];

   private final PropertyChangeSupport pcs = new
PropertyChangeSupport( this );

   public String getTitle()
   {
      return this.title;
   }

   public void setTitle( String title )
   {
      String old = this.title;
      this.title = title;
      this.pcs.firePropertyChange( "title", old, title );
   }

   public String[] getLines()
   {
      return this.lines.clone();
   }

   public String getLines( int index )
   {
      return this.lines[index];
   }

   public void setLines( String[] lines )
   {
      String[] old = this.lines;
      this.lines = lines;
      this.pcs.firePropertyChange( "lines", old, lines );
   }

   public void setLines( int index, String line )
   {
      String old = this.lines[index];
```

```
        this.lines[index] = line;
        this.pcs.fireIndexedPropertyChange( "lines", index, old, lines );
    }

    public void addPropertyChangeListener( PropertyChangeListener listener
)
    {
        this.pcs.addPropertyChangeListener( listener );
    }

    public void removePropertyChangeListener( PropertyChangeListener
listener )
    {
        this.pcs.removePropertyChangeListener( listener );
    }

    protected void paintComponent( Graphics g )
    {
        g.setColor( getForeground() );

        int height = g.getFontMetrics().getHeight();
        paintString( g, this.title, height );

        if ( this.lines != null )
        {
            int step = height;
            for ( String line : this.lines )
                paintString( g, line, height += step );
        }
    }

    private void paintString( Graphics g, String str, int height )
    {
        if ( str != null )
            g.drawString( str, 0, height );
    }
}
```

Constrained Properties:

```
import java.io.Serializable;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.beans.PropertyVetoException;
import java.beans.VetoableChangeListener;
import java.beans.VetoableChangeSupport;
import java.awt.Graphics;
import javax.swing.JComponent;


public class MyBean
        extends JComponent
        implements Serializable
{
    private String title;
    private String[] lines = new String[10];

    private final PropertyChangeSupport pcs = new
PropertyChangeSupport( this );
    private final VetoableChangeSupport vcs = new
VetoableChangeSupport( this );

    public String getTitle()
    {
        return this.title;
    }

    public void setTitle( String title )
            throws PropertyVetoException
    {
        String old = this.title;
        this.vcs.fireVetoableChange( "title", old, title );
        this.title = title;
        this.pcs.firePropertyChange( "title", old, title );
    }

    public String[] getLines()
    {
        return this.lines.clone();
    }

    public String getLines( int index )
```

```
   {
      return this.lines[index];
   }

   public void setLines( String[] lines )
         throws PropertyVetoException
   {
      String[] old = this.lines;
      this.vcs.fireVetoableChange( "lines", old, lines );
      this.lines = lines;
      this.pcs.firePropertyChange( "lines", old, lines );
   }

   public void setLines( int index, String line )
         throws PropertyVetoException
   {
      String old = this.lines[index];
      this.vcs.fireVetoableChange( "lines", old, line );
      this.lines[index] = line;
      this.pcs.fireIndexedPropertyChange( "lines", index, old, line );
   }

   public void addPropertyChangeListener( PropertyChangeListener listener
)
   {
      this.pcs.addPropertyChangeListener( listener );
   }

   public   void   removePropertyChangeListener(   PropertyChangeListener
listener )
   {
      this.pcs.removePropertyChangeListener( listener );
   }

   public   void   addVetoableChangeListener(   VetoableChangeListener
listener )
   {
      this.vcs.addVetoableChangeListener( listener );
   }

   public  void  removeVetoableChangeListener(  VetoableChangeListener
listener )
   {
      this.vcs.removeVetoableChangeListener( listener );
   }

   protected void paintComponent( Graphics g )
   {
      g.setColor( getForeground() );

      int height = g.getFontMetrics().getHeight();
      paintString( g, this.title, height );

      if ( this.lines != null )
      {
         int step = height;
         for ( String line : this.lines )
            paintString( g, line, height += step );
      }
   }

   private void paintString( Graphics g, String str, int height )
   {
      if ( str != null )
         g.drawString( str, 0, height );
   }
}
```

Announcement

Types of Java Bean properties:

Simple – A bean property with a single value whose changes are independent of changes in any other property.

Indexed – A bean property that supports a range of values instead of a single value.

Bound – A bean property for which a change to the property results in a notification being sent to some other bean.

Constrained – A bean property for which a change to the property results in validation by another bean. The other bean may reject the change if it is not appropriate

Summary

In this session, we learnt :

- About reusable software components
- Various technology alternatives in Java Beans
- Definition of a Java Bean
- Also We discussed the features of Java Bean
- Studies various Java Bean Properties

Assignment

1. Write Steps to Create Simple Visual Java Bean?

References

- Developing Java Beans , Robert Englander, Oreilly Publications
- http://www.roseindia.net/programming/tutorials/JavaBeansTutorials.shtml
- http://www.javapassion.com/javase/javabeans.pdf
- http://download.oracle.com/javase/tutorial/javabeans/