Unit-3

Java Beans

Introduction to Java Beans

- A java beans is a software component that has been designed to be reusable in a variety of different environments.
- Java beans are reusable software components.
- It may perform a simple function such as checking the spelling of documents or complex one.
- Beans are important because they allow you to build complex system from software component.
- These component may be provided by you or supplied by one or more different venders.
- Java beans defines an architectures that satisfies how these building blocks can operate together.

Introduction to Java Beans

- A simple java object must support the following features for it to become a Java Bean.
  - Customization
  - Communication
  - Persistence
  - Introspection

Customization

- It is the ability of JavaBeans to allow its properties to be changed during the build and execution phases.

Communication

- It is ability of JavaBeans to inform changes in its attributes to other beans and to the container application.

Introduction to Java Beans

Persistence

- It is the ability of java beans to save its state to a disk or any other storage device when the execution of container application that hold the beans is terminated.
- Persistence enables beans to save and restore their state.
- The JavaBeans architecture uses Java Object Serialization to support persistence.

Introspection

- It is the process of analyzing a bean to determine its capability.
- It is the ability of a beans to allow an external application to query the properties, methods, and events supported by it.

Design pattern for Properties

- A property is a subset of a Bean’s state. A bean property is named attribute of bean that can affect its behavior or appearance. Example of bean properties include color, label, font, font size and display size.
- Properties are the private data member of java bean classes.
- Properties are used to accept input from an end user in order to customize a java beans
- Types of JavaBeans Properties
  - **Simple Properties**
  - **Boolean Properties**
  - **Indexed Properties**

Simple Properties

- Simple properties refers to the private variables of JavaBean that can have only single value.
- Simple properties are retrieved and specified using the **get** and **set** methods respectively.
- The syntax of get and set method:

  Set method: public void setN(T arg)

  Get method: public T getN()

- Where N is the name of property and T is its type.

Boolean Properties:

- A Boolean property is a property which is used to represent the values true or false.

Syntax:

- Let N be the name of property and T be the type of value the

  public Boolean isN()

  public void setN(boolean arg)

  public Boolean getN()

- For getting values ***isN()*** and ***getN()*** methods are used and for setting the Boolean values ***setN()*** method is used.

Example:

```
public boolean dotted=false;

public boolean isDotted()

{

  return dotted;

}

public void setDotted(Boolean dotted)

{

   this.dotted=dotted;

}
```

Indexed Properties:

- Indexed properties are consist of multiple values.
- If a simple property can hold an array of value they are no longer called simple but instead indexed properties.
- The method's signature has to be adapted accordingly
- An indexed property may expose set/get methods to read/write one element in the array.
- Indexed properties enable you to set or retrieve the values from an array of property values.

Syntax:

public T getN(int index);

public void setN(int index, T value)

public T[] getN()

public void setN(T values[])

Here is an indexed property called data along with its getter and setter method:

```
private double data;
public double getData(int index){
  return data[index];
}
public void setData(int index, double value){
  data[index]=value;
}
```

Design Pattern for event

- Beans can generate event and send them to other objects.
- These can be identified by the following design patterns, where T is type of the event.

  public void addTListener(Tlistener eventListener)

  public void removeTListener(Tlistener eventListener)
- For example assuming an event interface tyoe called TemeratureListener, Bean that monitors temperature might supply the following method.

  public void addTemperatureListener(TemperatureListener t1)
  {…}
  public void removeTemperatureListener(TemperatureListener t1)
  {…}

Methods and Design Pattern

- Design pattern are not used for naming non-property methods.
- Introspection mechanism finds all of the public methods of bean.
- Protected and private methods are not presented.

Bounds and Constraints Properties

Bound properties

- When a bound property is changed, a property change event is multicast to other beans to inform them of this update.
- Properties of bean that provide notification to other beans about changes in its value are called bound properties.

Constrained properties

- Some properties of bean might need to be prevented from being changed by another bean, such properties that have to be stopped from being changed by another bean are called constrained properties

Types of JavaBeans

Control beans

- Control beans are used to create GUI components that can be plugged into any application.

Container beans:

- Container beans are used to hold other java beans .
- You can create a container bean by extending one of the swing container classes.

Invisible Runtime beans:

- These are used to create components that perform specific task in background of an application.

Accessor and Mutator Methods

- Accessor and mutator methods are used to make the properties of bean available to the outside world.
- A **get (accessor)** method allows the current value of a property to be read by external applications.
- A **set (mutator)** method allows the current value of the property to be changed by the external application

public String getLanguage() { return language;}

public void setLanguage(String txt) { language=txt; }

Advantage of using Java beans

- A Bean has all the benefits of java's "write-once, run-anywhere" paradigm.
- You can control which properties, events, and methods of bean are available to an application builder.
- A bean may be designed to operate correctly in different locales(environments), and therefore useful in global markets.
- Auxiliary software can be provided to help a person configuration a bean.
- The configuration settings of bean can be saved in persistent storage and restored at later time.
- A bean may register to receive events from other object and can generate events that are sent to other objects.

Class vs. Beans

- **JavaBeans** are classes that encapsulate many objects into a single object (the **bean**). They are serializable, have a zero-argument constructor, and allow access to properties using getter and setter methods.
- A **class** is nothing but a blueprint or a template for creating different objects which defines its properties and behaviors. **Java class** objects exhibit the properties and behaviors defined by its **class**. A **class** can contain fields and methods to describe the behavior of an object.

Bean Development Kit(BDK) and Bean Box

- BDK is provided separately from JDK.
- BDK uses the tool called Bean Box for Bean creation.
- Bean Box is generally used to test your java beans.
- Bean Box is considered a reference builder tool environment.
- It is not designed for building GUI application.
- You can create a java bean and then use the bean box to test that it runs properly.
- If a java bean runs properly in bean box, you can be sure that it works properly with other commercial builder tools.

When you start the bean box , you will see three windows.

ToolBox Window

- It display the java beans that are currently installed in the bean box.
- When the bean box start, it automatically loads its ToolBox with bean in jar files contained in the bean/jar directory.
- You can add additional beans such as your beans, to the ToolBox.

BeanBox Window

- Itself appears initially as empty window .
- Use this empty window sometime referred as a “form” by other builder tools, for building applications.

Properties Window

- It display the current properties for selected bean.
- If no bean is selected such as when you first start bean box, then the properties window display the bean box properties.
- Use the properties window to edit a bean’s properties.

Creating a New Java Bean

- To create a new beans you must follow the following steps.

1) Create a directory for you’re the new bean.
2) Create java source file(s).
3) Compile the source file(s).
4) Create manifest file
5) Generate a Jar file
6) Start BDK(or any Builder tool)
7) Test

Creating a New Java Bean

Create a Directory for the New Bean (You can use netbeans and create a new project for that)

Create the Source File for the New Bean

Any java class act as java beans if it follows given rule:-

- Class should be public and implements serializable interface
- Class should have public default constructor and properties should be private.
- Each bean property have getter and setter methods to get and set values of the properties.
- Java beans should be packaged in user defined package.

Compile the Source Code for the New Bean

Compile the source code to create a class file. Type the following:

javac <java bean name>.java.

Create a manifest (.mtf) file

- The manifest file contains a list of all the files that make up a java Bean
- Java Bean is recognized by target application by virtue of the Java Bean having a special entry in the manifest file.

  Name: MyButton.class

  Java-Bean: True

- A manifest file is saved with a .mft extension.

Create the JAR (.jar) file

- A JAR file is similar to zip file and contains all the class file of an application.
- A java bean is packaged into a JAR file for distribution
- The JAR file contains the manifest file and all other files such as the class files and picture files of the bean.
- The target application unpackaged the JAR file, read the manifest file first and then loads the all class files of the bean.
- A JAR file has a .jar extension.

Example : create MyButton.jar file

jar cfm Mybutton.jar MyButton.mft MyButton.class

Where c- for create a new archive file

f- specifies that the archive file name

m- specifies the manifest information from the manifest file.

Start the Bean Box.

- *CD to* c:\Program Files\BDK1.1\beanbox\.

  Then type "run".

- Load JAR into Bean Box by selecting "LoadJar..." under the File menu.

Test

- After the file selection dialog box is closed, change focus to the "ToolBox" window. You'll see “MyButton" appear at the bottom of the toolbox window.
- Select MyButton.jar.
- Cursor will change to a plus. In the middle BeanBox window, you can now click to drop in what will appear to be a colored rectangle.

Example: simple MyButton component that you may be use like a JButton in your application

```
import javax.swing.*;
public class MyButton extends JButton  implements Serializable{
   public MyButton()
   {
      super();
   }
}
```

//Target Application

```
import javax.swing.*;
public class MyTargetApp extends JFrame
{
   public MyTargetApp()
   {
      MyButton btn=new MyButton();
      add(btn);
      setVisible(true);
      setSize(400,500);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   public static void main()
   {
      new MyTargetApp();
   }
}
```

The Java Beans API

- The Java Beans functionality is provided by a set of classes and interfaces in the **java.beans** package

Interface	Description
AppletInitializer	Methods in this inter face are used to initialize Beans that are also applets.
BeanInfo	This inter face allows a designer to specify information about the proper ties, events, and methods of a Bean.
Customizer	This inter face allows a designer to provide a graphical user inter face through which a Bean may be configured.
DesignMode	Methods in this inter face determine if a Bean is executing in design mode.
ExceptionListener	A method in this inter face is invoked when an exception has occurred.
Proper tyChangeListener	A method in this inter face is invoked when a bound proper ty is changed.
Proper tyEditor	Objects that implement this inter face allow designers to change and display proper ty values.
VetoableChangeListener	A method in this inter face is invoked when a constrained proper ty is changed.
Visibility	Methods in this inter face allow a Bean to execute in environments where a graphical user inter face is not available.

TABLE 28-1 The Inter faces in java.beans

Class	Description
BeanDescriptor	This class provides information about a Bean. It also allows you to associate a customizer with a Bean.
Beans	This class is used to obtain information about a Bean.
DefaultPersistenceDelegate	A concrete subclass of PersistenceDelegate.
Encoder	Encodes the state of a set of Beans. Can be used to write this information to a stream.
EventHandler	Suppor ts dynamic event listener creation.
EventSetDescriptor	Instances of this class describe an event that can be generated by a Bean.
Expression	Encapsulates a call to a method that returns a result.
FeatureDescriptor	This is the superclass of the Proper tyDescriptor, EventSetDescriptor, and MethodDescriptor classes.
IndexedProper tyChangeEvent	A subclass of Proper tyChangeEvent that represents a change to an indexed proper ty.
IndexedProper tyDescriptor	Instances of this class describe an indexed proper ty of a Bean.
IntrospectionException	An exception of this type is generated if a problem occurs when analyzing a Bean.
Introspector	This class analyzes a Bean and constructs a BeanInfo object that describes the component.

MethodDescriptor	Instances of this class describe a method of a Bean.
ParameterDescriptor	Instances of this class describe a method parameter.
PersistenceDelegate	Handles the state information of an object.
Proper tyChangeEvent	This event is generated when bound or constrained proper ties are changed. It is sent to objects that registered an interest in these events and that implement either the Proper tyChangeListener or VetoableChangeListener inter faces.
Proper tyChangeListenerProxy	Extends EventListenerProxy and implements Proper tyChangeListener.
Proper tyChangeSuppor t	Beans that suppor t bound proper ties can use this class to notify Proper tyChangeListener objects.
Proper tyDescriptor	Instances of this class describe a proper ty of a Bean.
Proper tyEditorManager	This class locates a Proper tyEditor object for a given type.
Proper tyEditorSuppor t	This class provides functionality that can be used when writing proper ty editors.
Proper tyVetoException	An exception of this type is generated if a change to a constrained proper ty is vetoed.
SimpleBeanInfo	This class provides functionality that can be used when writing BeanInfo classes.
Statement	Encapsulates a call to a method.
VetoableChangeListenerProxy	Extends EventListenerProxy and implements VetoableChangeListener.

The BeanInfo Interface :

- The BeanInfo interface enables to explicitly control what information is available.
- The BeanInfo interface defines several method including these:

  PropertyDescriptor [] getPropertyDescriptor()

  EventSetdDescriptor[] getEventSetDescriptor()

  MethodDescriptor[] getMethodDescriptor()

- The class PropertyDescriptor, EventSetDescriptor and MethodDescriptor are defined within java.beans package.
- By implementing these method a developer can presented to a user, bypassing introspection based on design patterns.

```
package MyBean;
import java.beans.*;
public class DemoIntrospector {
    public static void main(String[] args) throws Exception {
        BeanInfo beanInfo = Introspector.getBeanInfo(Student.class);
         Student instance = (Student)
Beans.instantiate(DemoIntrospector.class.getClassLoader(),
                        beanInfo.getBeanDescriptor()
                                .getBeanClass()
                                .getName());

        System.out.println("The instance created : " + instance);
        BeanDescriptor bd = beanInfo.getBeanDescriptor();
        System.out.println("Bean name: " + bd.getName());
        System.out.println("Bean display name: " + bd.getDisplayName());
        System.out.println("Bean class: " + bd.getBeanClass());
```

```
for (PropertyDescriptor pd : beanInfo.getPropertyDescriptors()) {
        System.out.println("----------");
        System.out.println("Property Name: " + pd.getName());
        System.out.println("Property Display Name:" + pd.getDisplayName());
        System.out.println("Property Type: " + pd.getPropertyType());

        if (pd.getPropertyType()
          .isAssignableFrom(java.lang.String.class)) {
          System.out.println("Property value: " + pd.getReadMethod()
                                          .invoke(instance));
          pd.getWriteMethod()
            .invoke(instance, "a string value set");
          System.out.println("Property value after setting: " +
pd.getReadMethod()
.invoke(instance));
        }   }
}}
```