Unit-3

JavaBeans

Asst. Prof. Roshan Tandukar

JavaBean

- JavaBean is a portable, platform-independent component model written in the Java programming language
- With the JavaBeans API you can create reusable, platform-independent components
- Using JavaBeans-compliant application builder tools such as NetBeans or Eclipse, you can combine these components into applets, applications, or composite components
- JavaBean components are known as beans.
- Beans are dynamic in that they can be changed or customized
- Through the design mode of a builder tool, you use the property sheet or bean customizer to customize the bean and then save (persist) your customized beans.

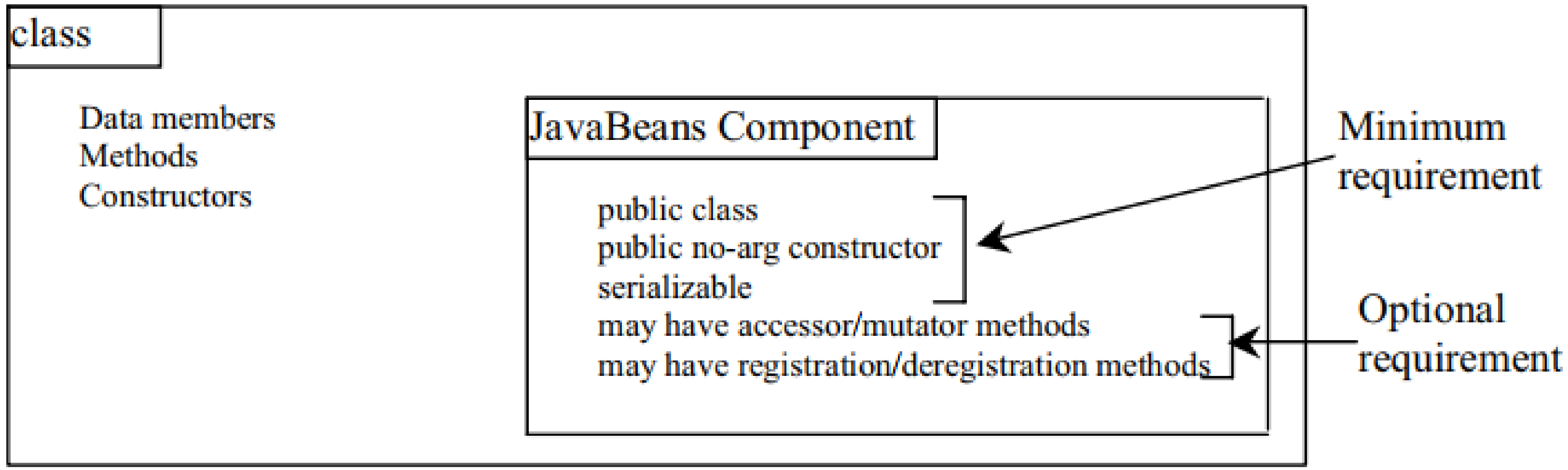
JavaBean

A JavaBeans component is a special kind of Java class that should be with the following requirements:

1. A bean must be a public class.
2. A bean must have a public no-arg constructor, though it can have other constructors if needed.
3. A bean must implement the **java.io.Serializable** interface to ensure a persistent state.
4. A bean usually has properties with correctly constructed public accessor(get) methods and mutator(set) methods that enable the properties to be seen and updated visually by a builder tool.
5. A bean may have events with correctly constructed public registration and deregistration methods that enable it to add and remove listeners. If the bean plays a role as the source of events, it must provide registration methods for registering listeners. For example, you can register a listener for **ActionEvent** using the **addActionListener** method of a **JButton** bean.

Note: The first three requirements must be observed, and therefore are referred to as minimum JavaBeans component requirements. The last two requirements depend on implementations.

JavaBean


class
Data members
Methods
Constructors
JavaBeans Component
public class
public no-arg constructor
serializable
may have accessor/mutator methods
may have registration/deregistration methods
Minimum requirement
Optional requirement

Advantages of Java Beans

1. A Bean obtains all the benefits of Java's "**write-once, run-anywhere**" paradigm.
2. The properties, events, and methods of a Bean that are exposed to another application can be controlled.
3. Auxiliary software can be provided to help configure a Bean. This software is only needed when the design-time parameters for that component are being set. It does not need to be included in the run-time environment.
4. The state of a Bean can be saved in persistent storage and restored at a later time.
5. A Bean may register to receive events from other objects and can generate events that are sent to other objects.

Basic bean concepts

- Introspection
- Properties
- Events
- Persistence
- Methods

Builder Tools & Introspection

- Builder tools discover a bean's features (that is, its properties, methods, and events) by a process known as introspection.
- Beans support introspection in two ways:

1. By adhering to specific rules, known as design patterns, when naming bean features
2. By explicitly providing property, method, and event information with a related bean information class.

Properties

- Properties are discrete, named attributes of a Java bean that can affect its appearance or behavior which are often data fields of a bean.
- Examples of bean properties:
    - Color, label, font, font size, and display size.
    - the JButton component has a property named text that represents the text to be displayed on the button.
- Private data fields are often used to hide specific implementations from the user and prevent the user from accidentally corrupting the properties. So, accessor and mutator methods are provided to let the user read and write the properties.
- Beans expose properties so they can be customized at design time

Properties

- Builder tools introspect on a bean to discover its properties and expose those properties for manipulation
- Customization is supported in two ways:
    a. by using property editors
    b. by using more sophisticated bean customizers

Events

- A bean may communicate with other beans use events
- An event is a signal to the program that something has happened which may be triggered by external user actions, such as mouse movements, mouse button clicks, and keystrokes, or by the operating system, such as a timer.
- An event object contains the information that describes the event.
- The Java event model consists of the following three types of elements:
  1. The event object
  2. The source object
  3. The event listener object
- A bean that is to receive events (a listener bean) registers with the bean that fires the event (a source bean)
- Builder tools can examine a bean and determine which events that bean can fire (send) and which it can handle (receive)

Persistence

- Persistence enables beans to save and restore their state which is achieved through object serialization
- Object serialization means converting an object into a data stream and writing it to storage.
- After changing a bean's properties, you can save the state of the bean and restore that bean at a later time with the property changes intact
- Any applet, application, or tool that uses that bean can then "reconstitute" it by deserialization. The object is then restored to its original state
- For example, a Java application can serialize a Frame window on a Microsoft Windows, the serialized file can be sent with e-mail to a Solaris machine, and then a Java application can restore the Frame window to the previous exact state

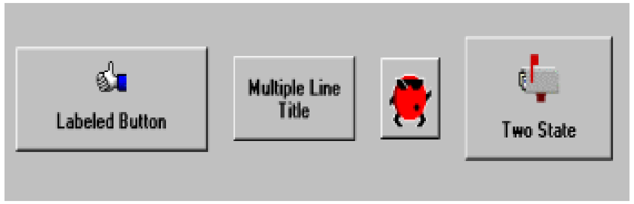
JavaBean Method

- A bean's methods are no different from Java methods, and can be called from other beans or a scripting environment
- By default all public methods are exported

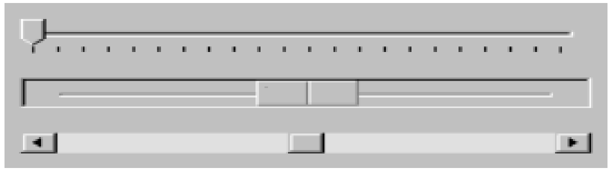
Examples of Beans

- GUI (graphical user interface) component
- Non-visual beans, such as a spelling checker
- Animation applet
- Spreadsheet application

- Button Beans



- Slider Bean

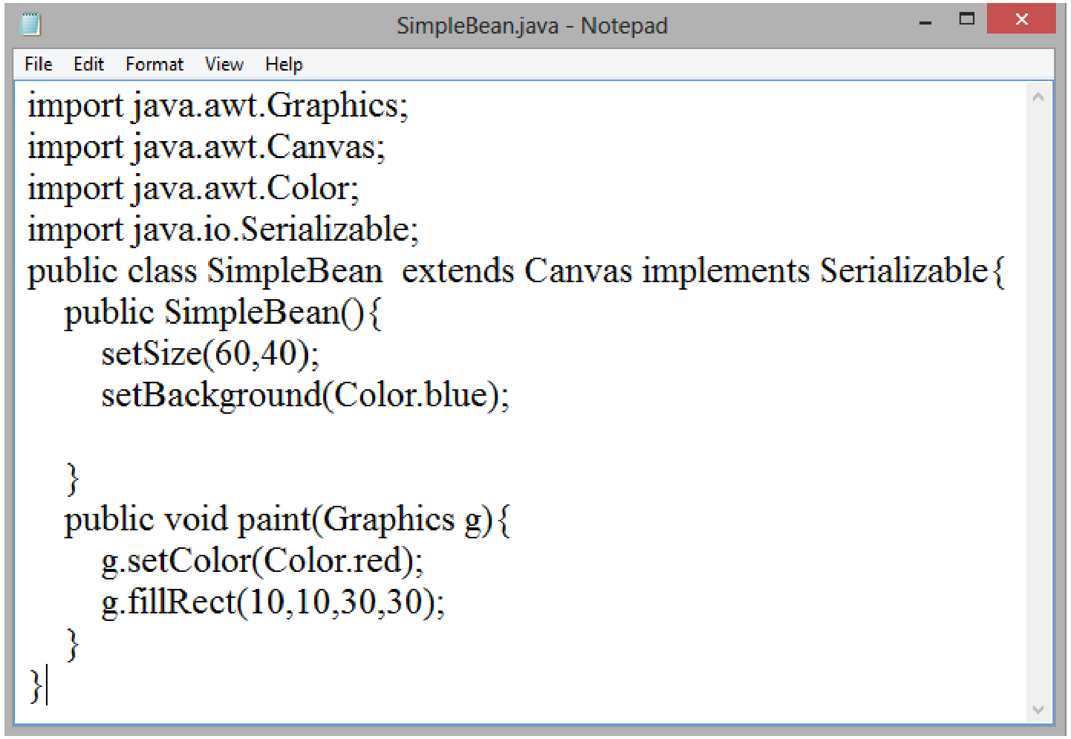


Creating a Simple Bean

1. Creating a simple Bean
2. Compiling and saving the Bean into a Java Archive (JAR) file
3. Loading the Bean into the Pallate of NetBeans
4. Dropping a Bean instance into the Netbean application
5. Inspecting the Bean's properties, methods, and events

Creating a Simple Bean

Creating a simple Bean



```
import java.awt.Graphics;
import java.awt.Canvas;
import java.awt.Color;
import java.io.Serializable;
public class SimpleBean  extends Canvas implements Serializable{
    public SimpleBean(){
        setSize(60,40);
        setBackground(Color.blue);

    }
    public void paint(Graphics g){
        g.setColor(Color.red);
        g.fillRect(10,10,30,30);
    }
}
```

Creating a Simple Bean

Compile the Bean

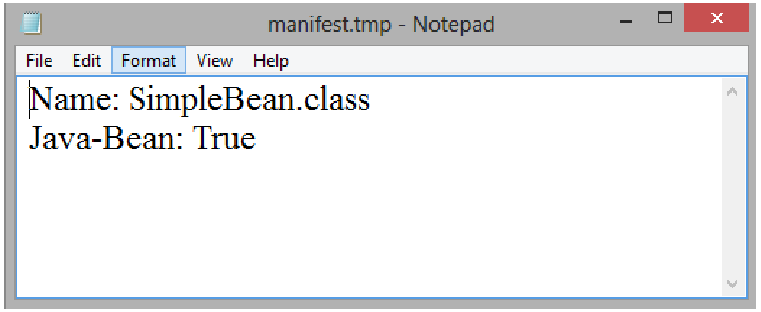
```
javac SimpleBean.java
```

- This produces the class file SimpleBean.class

Create a manifest file

- Use a text editor(eg. notepad) to create a file, and name it manifest.tmp, that contains the following text:

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



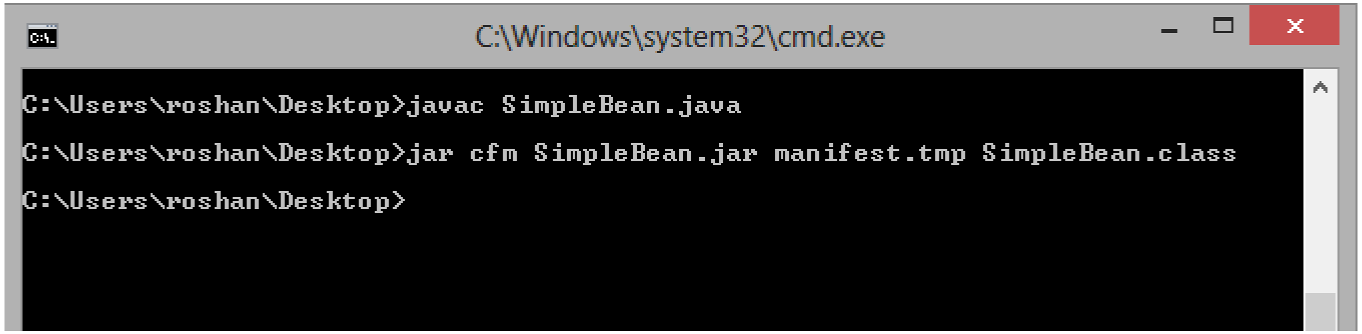
```
Name: SimpleBean.class
Java-Bean: True
```

Creating a Simple Bean

Create the JAR file

- The JAR file will contain the manifest and the SimpleBean class file:

  jar cfm SimpleBean.jar manifest.tmp SimpleBean.class



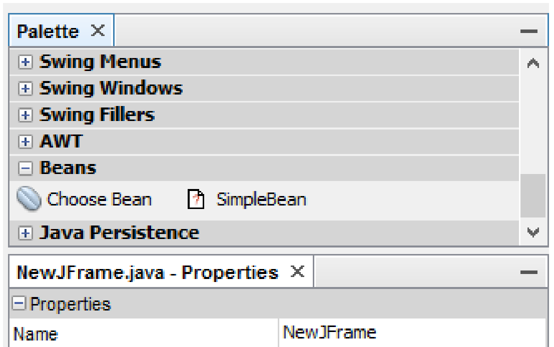
```
C:\Users\roshan\Desktop>javac SimpleBean.java

C:\Users\roshan\Desktop>jar cfm SimpleBean.jar manifest.tmp SimpleBean.class

C:\Users\roshan\Desktop>
```

c-create new archieve,
f-specify archieve file name,
m-include manifest information from the specified manifest file

Creating a Simple Bean

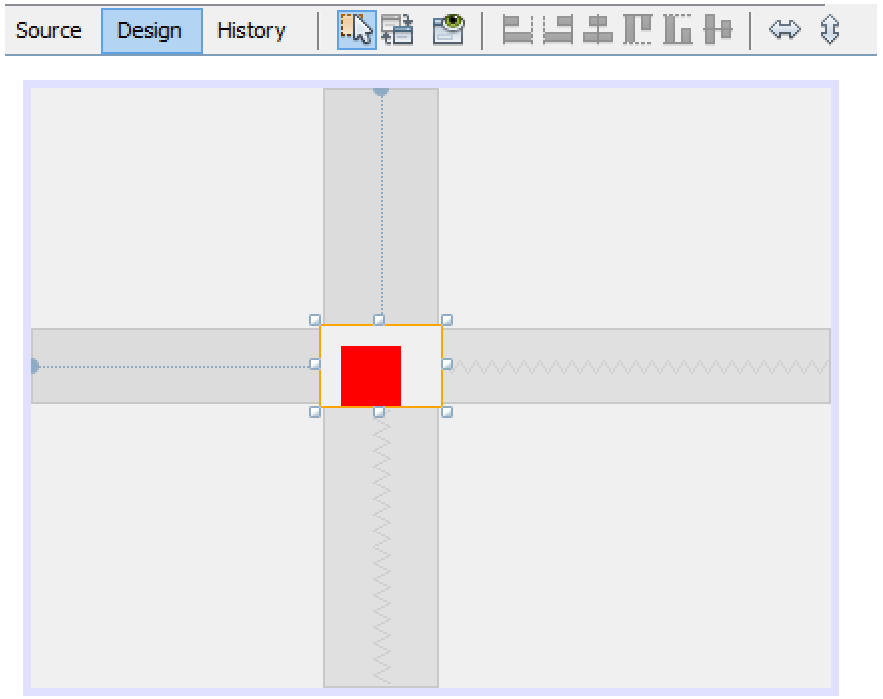
Load the JAR file into the Netbeans palatte

- To add the bean to the NetBeans palette, choose Tools > Palette > Swing/AWT Components from the NetBeans menu.
- Click on the Add from JAR... button.
- NetBeans asks you to locate the JAR file that contains the beans you wish to add to the palette. Locate the file you just downloaded and click Next.
- NetBeans shows a list of the classes in the JAR file. Choose the ones you wish you add to the palette. In this case, select SimpleBean and click Next.
- Choose Beans and click Finish.



Creating a Simple Bean

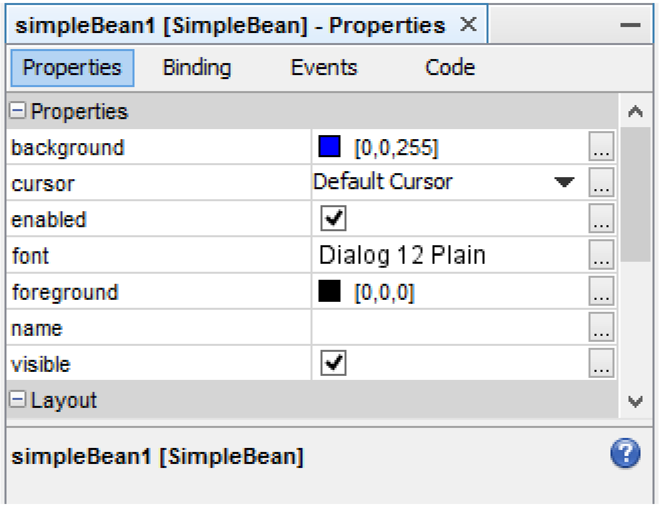
Drag and drop a SimpleBean from Netbeans palette into the the frame



Creating a Simple Bean

Inspecting the Bean's properties, methods, and events

- The Properties sheet displays the selected Bean's properties.
- We declared no properties in SimpleBean, so these are properties inherited from Canvas.



simpleBean1 [SimpleBean] - Properties

Properties | Binding | Events | Code

Properties

background	🟦 [0,0,255]
cursor	Default Cursor
enabled	✓
font	Dialog 12 Plain
foreground	■ [0,0,0]
name	
visible	✓

Layout

simpleBean1 [SimpleBean]

Types of 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.

Bound and Constrained Properties

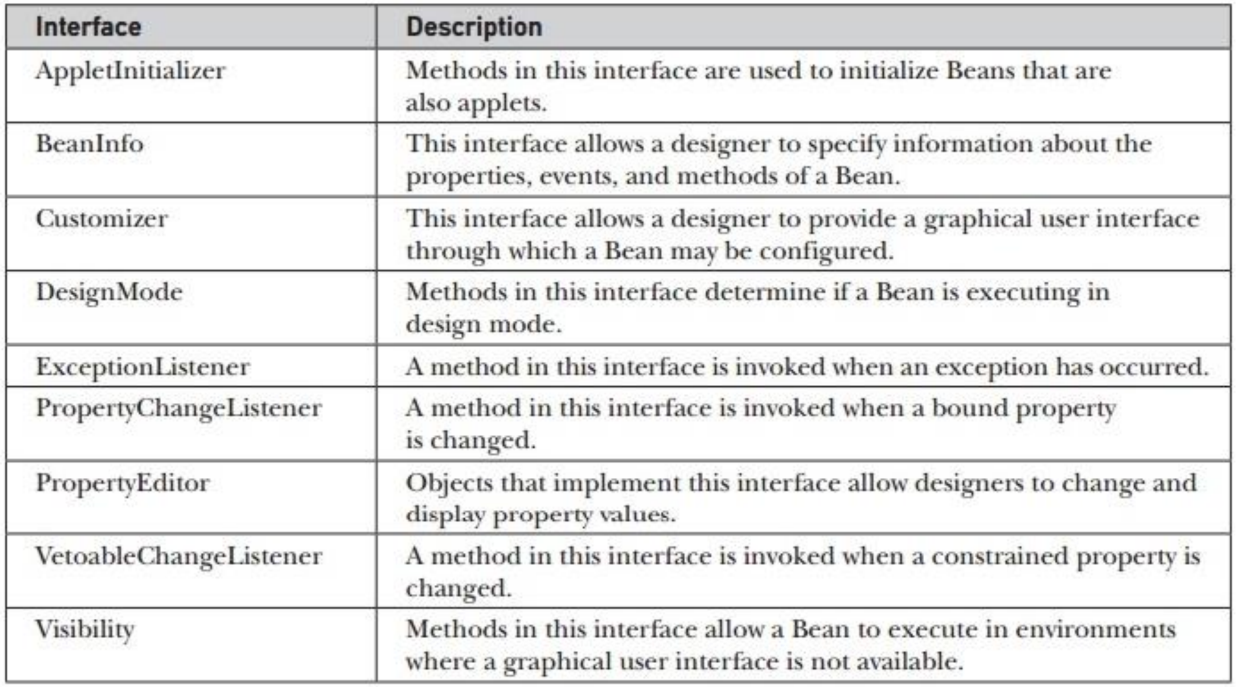
- A Bean that has a bound property generates an event when the property is changed.
- The event is of type PropertyChangeEvent and is sent to objects that previously registered an interest in receiving such notifications.
- A class that handles this event must implement the PropertyChangeListener interface.
- A Bean that has a constrained property generates an event when an attempt is made to change its value.
- It also generates an event of type PropertyChangeEvent.
- It too is sent to objects that previously registered an interest in receiving such notifications. However, those other objects have the ability to veto the proposed change by throwing a PropertyVetoException.
- This capability allows a Bean to operate differently according to its run-time environment. A class that handles this event must implement the VetoableChangeListener interface.

Persistence

- Persistence is the ability to save the current state of a Bean, including the values of a Bean's properties and instance variables, to nonvolatile storage and to retrieve them at a later time.
- The object serialization capabilities provided by the Java class libraries are used to provide persistence for Beans.
- The easiest way to serialize a Bean is to have it implement the java.io.Serializable interface, which is simply a marker interface.
- Implementing java.io.Serializable makes serialization automatic.
- Automatic serialization can also be inherited. Therefore, if any superclass of a Bean implements java.io.Serializable, then automatic serialization is obtained.
- If a Bean does not implement java.io.Serializable, you must provide serialization yourself, such as by implementing java.io.Externalizable. Otherwise, containers cannot save the configuration of your component.

Java Beans API

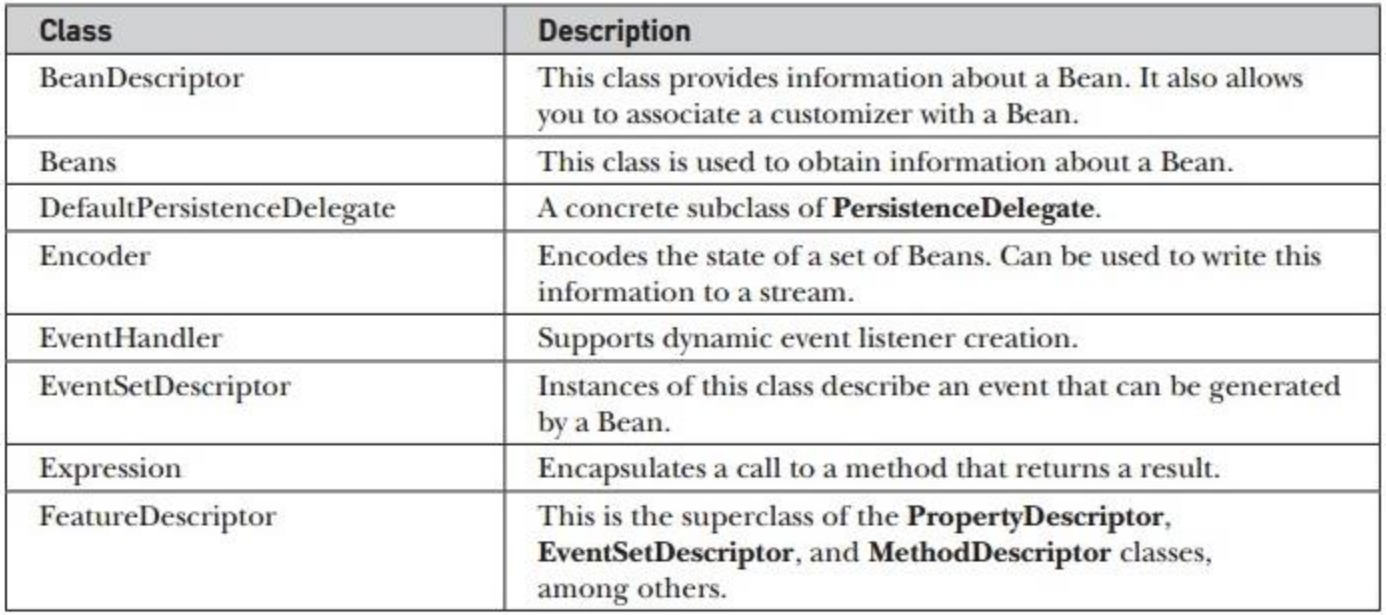
- The Java Beans functionality is provided by a set of classes and interfaces in the java.beans package.



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

Java Beans API

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



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	Supports 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 **PropertyDescriptor**, **EventSetDescriptor**, and **MethodDescriptor** classes, among others.

Customizers

- A Bean developer can provide a customizer that helps another developer configure the Bean.
- A customizer can provide a step-by-step guide through the process that must be followed to use the component in a specific context.
- A Bean developer has great flexibility to develop a customizer that can differentiate his or her product in the marketplace.

BeanInfo interface

- Use the BeanInfo interface to create a BeanInfo class and provide explicit information about the methods, properties, events, and other features of your beans.
- When developing your bean, you can implement the bean features required for your application task omitting the rest of the BeanInfo features.
- They will be obtained through the automatic analysis by using the low-level reflection of the bean methods and applying standard design patterns.
- You have an opportunity to provide additional bean information through various descriptor classes.

BeanInfo interface

- The BeanInfo interface enables you to explicitly control what information is available.
- The BeanInfo interface defines several methods, including these:
  - PropertyDescriptor[ ] getPropertyDescriptors( )
  - EventSetDescriptor[ ] getEventSetDescriptors( )
  - MethodDescriptor[ ] getMethodDescriptors( )
- They return arrays of objects that provide information about the properties, events, and methods of a Bean.
- The classes PropertyDescriptor, EventSetDescriptor, and MethodDescriptor are defined within the java.beans package, and they describe the indicated elements.
- By implementing these methods, a developer can designate exactly what is presented to a user, bypassing introspection based on design patterns.

Creating a Simple Bean

```
import java.beans.BeanInfo;
import java.beans.IntrospectionException;
import java.beans.Introspector;
import java.beans.PropertyDescriptor;
import java.io.Serializable;
public class SimpleBean  implements Serializable{
   private final String name="SimpleBean";
   private int size;
   public String getName(){
      return this.name;
   }
   public int getSize(){
      return this.size;
}
public void setSize(int size){
     this.size=size;
}
public static void main(String args[]) throws IntrospectionException{
      BeanInfo info=Introspector.getBeanInfo(SimpleBean.class);
      for(PropertyDescriptor pd: info.getPropertyDescriptors())
         System.out.println(pd.getName());
   }
}
```