JAVA BEANS

What are Java Beans?

1. A Java Bean is a software component that is designed to be reusable.
2. It is a java class that has a zero-argument (empty) constructor. So we can either not specify a constructor at all, or define a constructor with no arguments.
3. JavaBean class has no public instance variables (fields). Therefore we have to use the accessor methods instead of allowing direct access to the private fields.
4. Values are accessed through methods called as *getXXX* and *setXXX*. If a class has a method *getTitle* that returns a string, the class is said to have a String property named *Title*.
5.
6. A bean may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio.
7. A Bean may be visible to an end user, e.g., a button on a graphical user interface. A Bean may also be invisible to a user, e.g., software to decode a stream of multimedia information in real time.
8. A Bean may be designed to work autonomously on a user's workstation or to work in cooperation with a set of other distributed components. E.g., software to generate a pie chart from a set of data points is an example of a Bean that can execute locally. However, a Bean that provides real-time price information from a stock will have to work in cooperation with other distributed software to obtain its data.

What are the advantages of using Java beans?

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

Explain the term "introspection" in the context of Java Beans.

1. Introspection is the process of analyzing a Bean to determine its capabilities. This is an important feature of the Java Beans API, because it allows an application builder tool to obtain information about a component.
2. There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by an application builder tool.
3. In the first method, simple naming conventions are used. These allow the introspection mechanisms to infer information about a Bean.
4. In the second way, an additional class is provided that explicitly supplies this information.
5. **Design Pattern for Properties:**
    a. A design pattern specifies the rules for forming the method signature, return type and even its name. Design patterns can provide a useful documentation hint for programmers.
    b. The major design patterns are:
        i. Property design pattern,
        ii. Event design pattern, and
        iii. Method design pattern

c. A property is a named attribute. It specifies the characteristic, behavior and state of the object. Properties are functions which can get or set the values of a variable. A property is a subset of a Bean's state. The values assigned to the properties determine the behavior and appearance of the component.

d. Property design patterns are used to identify the publicly accessible properties of a bean.

e. The *setter* method is used to set a property and the *getter* method is used to obtain the property.

f. If a property has only a getter method, JavaBeans assumes that it is a read-only property. If both getter and setter methods are defined, JavaBeans assumes it is a read-write property.

g. There are three types of properties: *simple, Boolean* and *indexed*.

h. **Simple Properties:**
A simple property has a single value. The following design pattern is used:
public T getN()
public void setN(T arg)
Here **N** is the name of the property and **T** is its type.

A read/write property has both these methods to access its value. A read-only property has only the get method. The write-only property has only the set method.

Example:

```
private double depth, height, width;

public double getDepth()
{
    return depth;
}

public double setDepth(double d)
{
    depth = d;
}

public double getHeight()
{
    return height;
}

public double setHeightdouble h)
{
    height = h;
}

public double getWidth()
{
    return width;
}

public double setWidth(double w)
{
    width = w;
}
```

i. **Boolean Properties:**
The design pattern is as follows:
public Boolean isN( );
public void setN(Boolean a);

j. **Indexed Properties:**
An indexed property consists of multiple values, i.e,, it is an array of simple properties. It has the following design pattern:

public T getN (int *index*);
public void setN (int *index*, T *value*);
public T [ ] getN ();
public void setN (T *values*[]);

Consider an indexed property **data**. Its getter and setter methods are shown below:
private double data [ ];

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

```
public double [ ] getData()
{
    return data;
}

public void setData(double [ ]
values)
{
data = new double[values.length];
System.arraycopy(values, 0, data, 0,
values.length);
}
```

6. Design Pattern for Events:
When a Bean's internal state changes, it is often necessary to inform the other object of the change. Beans can do this by communicating events with other objects.

Beans send event notifications to event listeners that have registered themselves with a bean. Beans use the delegation event model. An event is generated by the bean and sent to other objects.

```
public void addTListener(TListener eventListener);
public void removeTListener(TListener eventListener);
```

These methods are used to add or remove a listener for the specified event.

For example, to add support for event listener that implements the ActionListener interface, we write:

```
public void addActionListener(ActionListener al);
public void removeActionListener(ActionListener al);
```

TheActionListener interface is used to receive actions like mouse clicks.

7. Constrained Bean Property:
A Bean property is constrained when any change to that property can be prevented. The Bean can itself prevent a property change.

The 3 parts of a constrained property implementation are:
- A source bean containing constrained properties.
- Listener object that can accept or reject proposed chages to the constrained property in the source bean.
- A PropertyChangeEvent object containing the property name, its old value and new values.

8. Bound Bean Property:
A Bean that has a bound property generates an event when the property is changed. The event is of the type **PropertyChangeEvent** and it is sent to the objects that have registered for this event notification.

What is Persistence in the context of JavaBeans?

Persistence is the ability to save a Bean to nonvolatile storage (disk) and retrieve it at a later time. Configuration settings are important and are saved this way.

Java class libraries have object serialization capabilities. Serialization is the process of writing the state of an object to a byte stream (file). A Bean must implement the **java.io.Serializable** interface. This makes serialization automatic.

Write short note on JAR files? What are the options available with JAR file?

1. JAR – Java Archive
2. A Jar file has the extension .jar.
3. It is a compressed file that contains classes and other resources which are required to use beans. The Jar files can contain videos files, image files, audio files, etc.,
4. BDK expect Beans to be packaged within JAR files.
5. A JAR file allows you to efficiently deploy a set of classes and their associated resources. For example, a developer may build a multimedia application that uses various sound and image files. A set of Beans can control how and when this information is presented. All of these pieces can be placed into one JAR file.
6. The elements in a JAR file are compressed and so downloading a JAR file is faster than downloading several uncompressed files.
7. Digital signatures can also be associated with the individual elements in a JAR file. This allows a consumer to be sure that these elements were produced by a specific organization or individual.

The jar utility is : jar *options file*

Example: Create a JAR file named XYZ.Jar that contains all the .class and .gif files in the current directory.

```
jar cf Xyz.jar  *.class  *.gif
```

<u>Options available with jar:</u>

Option	Description
c	A new archive is to be created.
C	Change directories during command execution.
f	The first element in the file list is the name of the archive that is to be created or accessed.
i	Index information should be provided.
m	The second element in the file list is the name of the external manifest file.
M	Manifest file not created.
t	The archive contents should be tabulated.
u	Update existing JAR file.
v	Verbose output should be provided by the utility as it executes.
x	Files are to be extracted from the archive.
0	Do not use compression.

Examples:

Tabulating the Contents of a jar file:

```
jar tf Xyz.jar
```

Extracting files from a jar file:

```
jar xf Xyz.jar
```

Updating an existing jar file:

```
jar -uf Xyz.jar file1.class
```

What is the manifest file?

1. A manifest file is a text file which contains the descriptions of the beans contained in the JAR files.
2. A manifest file indicates which of the components in a JAR file are Java Beans. An example of a manifest file is this: The JAR file contains four .gif files and one .class file. The last entry is a Bean.

   Name: slide0.gif
   Name: slide1.gif
   Name: slide2.gif
   Name: slide3.gif
   Name: Slides.class
   Java-Bean: True
3. A manifest file may reference many.class files. If a .class file is a Java Bean, its entry must be immediately followed by the line "Java-Bean: True".

Explain the BeanInfo interface.

1. There are two ways to determine what information is available to the user of a Bean.
2. One way is through the **design patterns**. The design pattern implicitly determines what properties are available to the user.
3. The other way is by using the **BeanInfo** interface.
4. The BeanInfo interface allows the programmer to explicitly control what information is available.
5. The methods of BeanINfo interface are:

   PropertyDescriptor [] getPropertyDescriptors()
   EventSetDescriptor [] getEventSetyDescriptors()
   MethodDescriptor [] getMethodDescriptors()
6. All these methods return an *array* of objects that provide information about properties, events and methods of a Bean.
7. The classes PropertyDescriptor, EventSetDescriptor, and MethodDescriptor are defined in the **java.beans** package.
8. When creating a class that implements BeamInfo, we must give it the name *bname*BeamInfo, where *bname* is the name of the Bean. E.g., MyBeanBeanInfo.

Discuss the following classes of JavaBeans API: Introspector, PropertyDescriptor, EventSetDescriptor, and MethodDescriptor

Introspector Class: This class provides methods that support introspection. The most important one is the getBeanInfo() method. This method returns a **BeanInfo** object that can be used to obtain information about the Bean.

PropertyDescriptor Class: This class describes the Bean properties. It has many methods that can manage and describe properties. E.g., we can determine if a property is bound by calling the **isBound**() method. Similarly there is a method **isConstrained**(). To get the name of the property we use the getName() method.

EventSetDescriptor: This class represents a Bean event. Methods of this class are: getAddListenerMethod() – this is used to obtain the methods used to add listeners; the getRemoveListenerMethod() is used to obtain methods used to remove listeners.

MethodDescriptor: This class represents a Bean method. We can use the method getName() to obtain the name of a method.

PROGRAMS

1. Write JavaBean program that finds square and cube value in terms of properties.

```
public class mathcalc
{
	int a;

	public mathcalc() { }

	public void setA(int m)
	{
		a = m;
	}

	public int getA()
	{
		return (a);
	}

	public int getSquare()
	{
		return (a * a);
	}

	public void setSquare( int m)		{ }

	public int getCube()
	{
		return (a * a * a);
	}

	public void setCube( int m)		{ }
```

2. Write JavaBean program to find the sum, difference, product and ratio of two numbers as properties.

```
public class cal
{
	int a, b;

	public cal()
	{
		a = 0; b = 1;
	}

	public int getA()
	{
		return a;
	}
```

```
	public void setA(int m)
	{
		a = m;
	}

	public int getB()
	{
		return b;
	}

	public void setB(int n)
	{
		b = n;
	}
```

```
public int getAdd()
{
      return (a + b);
}

public void setAdd(int m){ }

public int getSub()
{
      return (a - b);
}

public void setSub(int m) { }
```

```
      public int getMul()
      {
            return (a * b);
      }

      public void setMul(int m) { }

      public int getDiv()
      {
            return (a / b);
}

      public void setDiv(int m) { }
```

3. Write JavaBean program to find the reverse of a number as a property.

```
public class revno
{
      int n;

      public revno() {}

      public int getN()
      {
            return n;
      }

      public void setN(int m)
      {
            n = m;
      }

      public int getRevNo()
      {
            int d, revnum = 0;
            int temp = n;

            while (temp > 0)
            {
                  d = temp % 10;
                  revnum = (revnum * 10) + d;
                  temp = temp / 10;
            }

            return revnum;
      }

      public void setRevNum(int m){}
}
```

4. Write multiline JavaBean program.
In this program we use TextArea for multiline text.

```
import java.awt.*;
import java.awt.event.*;

public class multiline extends Panel
{
      int height, width;
      TextArea ta;

      public multiline()
      {
            TextArea ta = new TextArea(5, 50);

            add(ta);

            ta.setText("Learning Java Beans" +
                        "\n From Internet resources" +
                        "\n and books");
            setSize(300, 300);
            setVisible(true);
      }

      public int getheight()
      {
            return height;
      }

      public void setheight(int h)
      {
            height = h;
            setSize(width, height);
      }

      public int getwidth()
      {
            return width;
      }

      public void setwidth(int w)
      {
            width  = w;
            setSize(width, height);
      }
}
```