What is Javabean?

JavaBeans Concepts

- A component is a self-contained reusable software unit
- Components expose their features (public methods and events) to builder tools
- A builder tool maintains Beans in a palette or toolbox.
- You can select a bean from the toolbox, drop it in a form, and modify its appearance and behaviour.
- Also, you can define its interaction with other beans
- ALL this without a line of code.

JavaBean Characteristics

- a public class with 0-argument constuctor
- it has properties with accessory methods
- it has events
- it can customized
- its state can be saved
- it can be analyzed by a builder tool

Key Concepts

- A builder tool discover a bean's features by a process known as *introspection*.
  - Adhering to specific rules (design pattern) when naming Bean features.
  - Providing property, method, and event information with a related Bean Information class.
- Properties (bean's appearance and behavior characteristics) can be changed at design-time.
- Properties can be customized at design-time. Customization can be done:
  - using property editor
  - using bean customizers
- Events are used when beans want to intercommunicate
- Persistence: for saving and restoring the state
- Bean's methods are regular Java methods.

Javabean

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.
Following are the unique characteristics that distinguish a JavaBean from other Java classes:

- It provides a default, no-argument constructor.
- It should be serializable and implement the Serializable interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

Explain the advantages of Javabean.

Advantage of Java Bean

- **Write once, run anywhere**
- The properties, events, and methods of a bean that are exposed to an application builder tool can be controlled
- They are the interface of the bean.
- They are platform independent
- Configuration setting of a bean can be saved in persistent storage and restored later
- Bean may register and receive events from other object and can generate event sent to other objects (Bean communication)

What is Javabean property? Explain different types of properties.

JavaBeans Property:
A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including classes that you define.

A JavaBean property may be read, write, read only, or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class:

Method	**Description**
getPropertyName()	For example, if property name is firstName, your method name would be getFirstName() to read that property. This method is called accessor.
setPropertyName()	For example, if property name is firstName, your method name would be setFirstName() to write that property. This method is called mutator.

A read-only attribute will have only a getPropertyName() method, and a write-only attribute will have only a setPropertyName() method.

Properties

- ➢ Bean's appearance and behavior - changeable at design time.
- ➢ They are private values
- ➢ Can be accessed through getter and setter methods
- ➢ getter and setter methods must follow some rules -- design patterns (documenting experience)
- ➢ A builder tool can:
  - discover a bean's properties
  - determine the properties' read/write attribute
  - locate an appropriate "property editor" for each type

  - display the properties (in a sheet)
  - alter the properties at design-time
- Properties are Bean’s behavior and appearance attributes that can be changed at design time.
- Properties exposed for customization
- Customization by Property Editors or more custom classes

Types of Properties

- Simple
- Indexed: multiple-value properties
- Bound: provide event notification when value changes
- Constrained: how proposed changes can be okayed or vetoed by other object

Simple Properties

- When a builder tool introspect your bean it discovers two methods:
  - public Color getColor()
  - public void setColor(Color c)
- The builder tool knows that a property named “Color” exists -- of type Color.
- It tries to locate a property editor for that type to display the properties in a sheet.
- Adding a Color property
  - Create and initialize a private instance variable
    - private Color color = Color.blue;
  - Write public getter & setter methods

```
public Color getColor()
{
        return color;
}
public void setColor(Color c)
{
        color = c;
        repaint();
}
```

Property Editors

- A property editor is a user interface for editing a bean property. The property must have both, read/write accessor methods.
- A property editor must implement the *PropertyEditor* interface.
- *PropertyEditorSupport* does that already, so you can extend it.
- If you provide a custom property editor class, then you must refer to this class by calling *PropertyDescriptor.setPropertyEditorClass* in a *BeanInfo* class.
- Each bean may have a BeanInfo class which customizes how the bean is to appear. *SimpleBeanInfo* implements that interface

What is a JavaBean?
"A JavaBean is a reusable software component that can be manipulated visually in a builder tool"

JavaBeans Objectives

- Provide a platform neutral component architecture
- Simple to develop
- Leverage existing Java technologies
- Provide design-time support for visual builder tools

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

Property Types

- Simple Properties
- Bound Properties
- Indexed 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();
```

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);
```

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);
```

Constrained Properties

Allow registered listeners to validate a proposed change

Listeners must implement the *java.beans.VetoablecChangeListener* interface

vetoableChange(PropertyChangeEvent event) throws PropertyVetoException;

Javabean Properties

To define a property in a bean class, supply public getter and setter methods.

Various specializations of basic properties are available and described in the following

Indexed Properties

An *indexed* property is an array instead of a single value. In this case, the bean class provides a method for getting and setting the entire array.

Bound Properties

A *bound* property notifies listeners when its value changes.

Constrained Properties

A *constrained* property is a special kind of bound property. When a constrained property is about to change, the listeners are consulted about the change.

Overview of Event Model

- Provides a notification mechanism between a source object and one or more listener objects
- Source or listener objects does not need to be graphical components
- Supports introspection
- Extensible

Packaging and Deployment

- JAR (Java ARchive) files are the primary method of packaging and distributing JavaBeans
- A JAR file contains:
    - Class files
    - Serialized prototypes of a bean
    - Documentation files
    - Resources