Week 7: Introduction to JavaBeans

Overview of JavaBeans

JavaBeans are reusable software components written in the Java programming language. They follow a set of conventions that make them easy to create, use, and customize within Java applications. Think of JavaBeans as modular building blocks for Java programs. They can represent anything, from simple data structures (like a Person or Car JavaBean) to more complex components used in graphical user interfaces (GUIs), such as buttons, text fields, or custom UI elements.

Why Use JavaBeans?

JavaBeans provide a consistent and standardized way to build reusable components that can be manipulated visually within development tools like **NetBeans** or **Eclipse**. This makes it possible for developers to quickly assemble applications without having to write everything from scratch.

Key Features of JavaBeans

1. Properties

Properties are attributes of a JavaBean that can be accessed and modified using **getter** and **setter** methods. These methods follow a naming convention (e.g., getName(), setName()) that allows tools and frameworks to recognize and interact with them easily.

- **Example**: If you have a Car JavaBean, typical properties might include make, model, and year.
    - You can **get** the make of the car using a method like getMake().
    - You can **set** the model of the car using a method like setModel().

Example JavaBean with Properties:

```
public class Car {
   private String make;
   private String model;
   private int year;

   // Public no-argument constructor
   public Car() {
      this.make = "Unknown";
      this.model = "Unknown";
      this.year = 0;
   }
```

```
    // Getter and Setter methods
    public String getMake() {
        return make;
    }
    public void setMake(String make) {
        this.make = make;
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
    public int getYear() {
        return year;
    }
    public void setYear(int year) {
        this.year = year;
    }
}
```

```java
public class Car {
    private String make;
    private String model;
    private int year;

    // Public no-argument constructor
    public Car() {
        this.make = "Unknown";
        this.model = "Unknown";
        this.year = 0;
    }

    // Getter and Setter methods
    public String getMake() {
        return make;
    }

    public void setMake(String make) {
        this.make = make;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }
}
```

Explanation:

- The Car JavaBean has private properties (make, model, year).
- The public getter and setter methods provide controlled access to these properties.

2. Events

JavaBeans can generate **events** and listen for them. This allows components to communicate and respond to user actions or other changes. For example, in a graphical user interface, clicking a button can trigger an event, and another component can respond to that event.

- **Event Source**: The object that generates the event.
- **Event Listener**: The object that listens for and responds to the event.
- **Event Object**: Contains information about the event.

Example: A JavaBean representing a button can generate an event when it is clicked, and an event listener can respond by executing a specific action.

3. Customization

JavaBeans can be customized through property editors and methods. This means developers can change how a JavaBean behaves or appears without modifying its underlying code. IDEs like **NetBeans** offer visual tools to customize JavaBeans properties quickly.

- **Example**: You can change the background color of a GUI button JavaBean using a property editor in an IDE.

Putting It All Together: Practical Example

Creating a Person JavaBean:

```
public class Person {
    private String name;
    private int age;

    // No-argument constructor
    public Person() {
        this.name = "Unknown";
        this.age = 0;
    }

    // Getter for name
    public String getName() {
        return name;
    }

    // Setter for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter for age
    public int getAge() {
        return age;
    }

    // Setter for age
    public void setAge(int age) {
        this.age = age;
    }
}
```

Using the Person JavaBean:

```
public class TestPerson {
    public static void main(String[] args) {
        // Create a Person JavaBean
        Person person = new Person();

        // Set properties
        person.setName("Alice");
        person.setAge(30);

        // Get properties
        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
    }
}
```

Creating a Simple JavaBean

A basic JavaBean is a public class with a public no-argument constructor and provides getter and setter methods for accessing private properties.

Example 1: Creating a Simple JavaBean

Code Explanation: This example creates a Student JavaBean with properties like name and age.

Code:

```
// Save as Student.java
package mybeans;
import java.io.Serializable;

public class Student implements Serializable {
   // Private properties
   private String name;
   private int age;

   // No-argument constructor
   public Student() {
      this.name = "Unknown";
      this.age = 0;
   }
   // Getter for name
   public String getName() {
      return name;
   }
   // Setter for name
```

```
    public void setName(String name) {
        this.name = name;
    }
    // Getter for age
    public int getAge() {
        return age;
    }
    // Setter for age
    public void setAge(int age) {
        this.age = age;
    }
    // Method to display student info
    public void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}
```

Lab Activity: Creating and Customizing JavaBeans

1. **Create a New JavaBean**:
    - Create a JavaBean with properties like title and price for a Book class.
    - Provide getter and setter methods.
2. **Customization**:
    - Add a method to display the book details.
    - Test the JavaBean in a separate class.

Week 8: JavaBeans Properties & Event Handling

In JavaBeans, **properties** are characteristics or attributes that define the state of the bean. For example, in a Person JavaBean, properties could be name, age, and gender. Properties allow JavaBeans to store data, which can be accessed and modified using **getter** and **setter** methods.

- **Private Variables**: Properties are usually defined as private variables, meaning they can only be accessed within the class itself. This encapsulation ensures that data is protected and cannot be directly modified by external classes.
- **Public Getter and Setter Methods**: To access and modify these properties, we use public methods:
    - **Getter Method**: Used to retrieve (or "get") the value of a property.
    - **Setter Method**: Used to change (or "set") the value of a property.

Example of a Simple Property in a JavaBean:

```
public class Person {
    // Private property
    private String name;

    // Public no-argument constructor
    public Person() {
        this.name = "Unknown";
    }

    // Getter for the name property
    public String getName() {
        return name;
    }

    // Setter for the name property
    public void setName(String name) {
        this.name = name;
    }
}
```

Explanation:

- The name property is a private variable.
- The getName() method retrieves the value of name.
- The setName(String name) method sets a new value for name.

This approach allows controlled access to the properties, which makes JavaBeans easy to use and customize.

Types of Properties in JavaBeans

1. **Simple Properties**:
    - A simple property is a single value, such as a String or int.
    - Accessed through a single pair of getter and setter methods.

Example:

```
private int age;

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}
```

2. **Indexed Properties:**

- An indexed property is like an array, where you can access individual elements using an index.
- Allows you to work with a collection of values.

Example:

```
private String[] favoriteFoods;

public String[] getFavoriteFoods() {
    return favoriteFoods;
}

public String getFavoriteFood(int index) {
    return favoriteFoods[index];
}

public void setFavoriteFoods(String[] favoriteFoods) {
    this.favoriteFoods = favoriteFoods;
}

public void setFavoriteFood(int index, String food) {
    this.favoriteFoods[index] = food;
}
```

3. **Bound Properties**:

- A bound property notifies listeners when its value changes. This is useful when you need other components to react to changes in a property's value.
- Implemented using the PropertyChangeSupport class, which manages listeners.

Example:

```
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;

public class Employee {
    private String position;
    private final PropertyChangeSupport support = new PropertyChangeSupport(this);

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        String oldPosition = this.position;
        this.position = position;
        support.firePropertyChange("position", oldPosition, position);
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        support.addPropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        support.removePropertyChangeListener(listener);
    }
}
```

1. **Explanation**:
    - The setPosition method fires a property change event whenever the position property is modified.
    - Listeners that are registered with the Employee bean will be notified of the change.

Understanding Events in JavaBeans

Events are mechanisms that allow JavaBeans to communicate with other components in an application. This communication is essential for building interactive applications where components can respond to user actions or other changes.

Key Concepts in Event Handling:

1. **Event Source**:
    - The object that generates an event (e.g., a button being clicked).
2. **Event Listener**:

- The object that listens for and responds to events. It must implement a listener interface (e.g., ActionListener).

3. **Event Object**:
    - Encapsulates the details about an event, such as the source of the event and any parameters associated with it.

Example of Event Handling in JavaBeans:

Imagine a button JavaBean that generates an event when it is clicked. Another component (the listener) can listen for this event and perform an action in response.

Code Example:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Example");
        JButton button = new JButton("Click Me");

        // Add an ActionListener to handle button clicks
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button was clicked!");
            }
        });

        frame.add(button);
        frame.setSize(200, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- The button is the **event source**.
- The ActionListener is the **event listener** that responds to the button click.
- When the button is clicked, an ActionEvent object is generated and passed to the listener's actionPerformed method.

Example 1: Customizing JavaBean Properties

Code Explanation: This example demonstrates creating a Book JavaBean with simple and bound properties.

Code:

```
// Save as Book.java
package mybeans;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.io.Serializable;

public class Book implements Serializable {
   private String title;
   private double price;
   private final PropertyChangeSupport pcs = new PropertyChangeSupport(this);
   // Constructor
   public Book(String title, double price) {
      this.title = title;
      this.price = price;
   }
   // Getter for title
   public String getTitle() {
      return title;
   }
   // Setter for title
   public void setTitle(String title) {
      String oldTitle = this.title;
      this.title = title;
      pcs.firePropertyChange("title", oldTitle, title);
   }
   // Getter for price
   public double getPrice() {
      return price;
   }
   // Setter for price
   public void setPrice(double price) {
      double oldPrice = this.price;
      this.price = price;
      pcs.firePropertyChange("price", oldPrice, price);
   }

   // Methods to add and remove property change listeners
   public void addPropertyChangeListener(PropertyChangeListener listener) {
      pcs.addPropertyChangeListener(listener);
```

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

Testing the Book JavaBean:

```
// Save as TestBook.java
package mybeans;
public class TestBook {
    public static void main(String[] args) {
        Book book = new Book("Java Programming", 29.99);
        book.addPropertyChangeListener(evt -> {
            System.out.println("Property " + evt.getPropertyName() + " changed from " +
                evt.getOldValue() + " to " + evt.getNewValue());
        });
        // Change properties
        book.setTitle("Advanced Java Programming");
        book.setPrice(39.99);
    }
}
```

Lab Activity: Implementing Event Handling in JavaBeans

1. **Create a Custom JavaBean**:
    - Create a Person JavaBean with properties like firstName and lastName.
    - Implement property change support.
2. **Implement Event Handling**:
    - Add property change listeners to respond to changes in firstName or lastName.

Exercises for Practice

1. **Exercise 1: Custom Properties** (15 Points)
    - Create a Car JavaBean with properties like make, model, and price.
    - Implement getter and setter methods.
    - Add event handling to listen for changes in any property.
2. **Exercise 2: Event Handling in JavaBeans** (15 Points)
    - Develop a Product JavaBean with a quantity property.
    - Use PropertyChangeSupport to fire an event whenever the quantity is updated.

- Write a class to test and display changes to the quantity property.