JavaBeans
The Component Model in Java

Mikhail Chalabine
mikch@ida.liu.se

JavaBeans is the component model for Java
- Portable
- Platform-independent
- Written in Java
- API introduced in Feb. 1997

A bean is a reusable software component
JavaBeans != Enterprise JavaBeans

Reference: JavaBeans Tutorial @ java.sun.com

Programming model

JavaBeans & Component Types

Visual components
- Used in Swing, AWT
- Visual application builders (visual composition)
  - Work flow: load, customize (size, color), save (persist)
  - Eclipse VEP (Visual Editor Project)
  - NetBeans

Non-visual components
- In Java2EE, Hibernate
- Capture business logic or state
A reusable component in Java

- Hides implementation, conforms to interfaces, encapsulates data
- Is written to a standard (component specification)
  - Implements the serializable interface (persistence)
  - No-argument constructor
    - E.g., instantiation through reflection
  - Design patterns or BeanInfo class (introspection)
  - Core features (methods, properties, events)

Beans’ core features

Properties

Events

Methods

Beans’ features: Methods

- Standard Java

Beans’ features: Properties (1)

- Appearance and behavior characteristics

```java
import java.io.Serializable;

public class MyJavaBean implements Serializable {
    private String first_name;
    private float income;
    ...
}
```

- Visual components
  - Builder tools can discover and expose
  - Customization - modifying appearance or behavior at design time by
    - Property editors (visual, programmable)
    - Bean customizers (visual, programmable)
Bean’s features: Properties (2)

- Specification suggests to have ‘getters’ and ‘setters’

```java
public String getFirst_name() {
    return first_name;
}
public void setFirst_name(String first_name) {
    this.first_name = first_name;
}
public float getIncome() {
    return income;
}
public void setIncome(float income) {
    this.income = income;
}
```

Bean’s features: Event model (1)

- **Fire (send) / handle (receive)**
  Components broadcast events and the underlying framework delivers the events to the components to be notified

- **Sources**
  - Define and fire events
  - Define methods for registering listeners

- **Listeners**
  - Get notified of events
  - Register using methods defined by sources

Bean’s features: Event model (2)

- **Write event class**
  - Create your own custom event class named `<NAME>Event` or use an existing event class, e.g., `ActionEvent`

```java
ActionEvent e;
```

- **Write event listener (handler, receiver)**
  - write `<NAME>Listener` interface and provide implementation of it or reuse existing listener interfaces, e.g., `ActionListener`
  - or complete handlers, so-called, adapters, e.g., `MouseAdapter`

```java
public class ButtonHandler implements ActionListener {
    ...
}
```

- **Write event source bean (Event generator)**
  - `JButton button = new JButton("Fire");`
  - In your custom bean implement `add<NAME>Listener()` and `remove<NAME>Listener()` methods. Implemented in JButton.

- **Register event listener**

```java
button.addActionListener( new ButtonHandler());
```
Example: Alarm Clock

- Properties
  - Current time
  - Alarm time
  - Alarm status (set/not set)
- Events
  - Alarm (source)

Example cont.

```
class AlarmClock implements Serializable {
    public AlarmClock() {...}
    public boolean getAlarmStatus() {...}
    public void setAlarmStatus(boolean value) {
        ...
    }
    ...
}
```

Optional class (study): AlarmClockCustomizer

```
interface AlarmListener {
    void alarmStarted(AlarmEvent e);
    void alarmStopped(AlarmEvent e);
}
```

```
CoffeeMachine
alarmStarted(AlarmEvent e)
alarmStopped(AlarmEvent e)
```

```
StereoSystem
alarmStarted(AlarmEvent e)
alarmStopped(AlarmEvent e)
```

Optional class (see below): AlarmClockInfo

Advanced features

- Optional class (see below): AlarmClockInfo
- Optional class (study): AlarmClockCustomizer
Discovering features through introspection

- We concluded that
  - For a bean to be the source of an event, it must implement methods to add and remove listener objects for that type of event
    
    ```java
    add<EventName>Listener(<EventName>Listener listener)
    remove<EventName>Listener(<EventName>Listener listener)
    ```

- For a bean to be the listener of an event, it must implement the `<EventName>Listener` interface

- We see that
  - 'add', 'remove', 'Listener', `<BeanClass>Info`, `<BeanClass>Customizer` form syntactic patterns

- We also said that
  - component specification suggests that bean properties should have setters and getters

Reminder (lecture on Java Reflection)

- Representing metalevel concepts at the base level is called reification.
- `/* Instantiate a metaobject */` Robot ourRobot = new Robot(...);
- `/* Obtain its (meta) class */` Class rClass = ourRobot.getClass();
- `rClass` represents (reflex) the class meta-level concept at the base level.

- Locate classes, methods, data accesses
- Allocate new classes, methods, fields
- Remove classes, methods, fields

Discovering beans’ features (1)

- Automatic (implicit)
  - adhering to design patterns makes a bean’s features discoverable through introspection

```java
import java.io.Serializable;

public class MyJavaBean implements Serializable {

    private String first_name;

    public String getFirst_name() {
        return first_name;
    }

    public void setFirst_name(String first_name) {
        this.first_name = first_name;
    }
}
```

Discovering beans’ features (2)

- Manual (explicit)
  - BeanInfo class (visual components)
    - Code that defines and initializes properties
      - Make properties visible / invisible, etc.
      - Expose / hide methods that the bean implements
        - setHidden() etc.
    - Use when
      - Bean code does not follow the standard naming convention (no introspection possible)
      - You intend to hide some features
Further topics

- Java techniques
  - Serialization and persistence in JavaBeans
- JavaBeans
  - More on customizers
  - More on properties

Evaluation

- Strengths
  - Simple - easy to use
  - Standard - mix vendors
  - Applicable for GUI development
- Weaknesses
  - Only suitable for GUI development
  - Not usable for non-programmers
  - Weak component market