

JavaBeans

Zaawansowane technologie Javy 2019

Komponenty programowe

- Ziarno to komponent programowy wielokrotnego użytku, oparty na specyfikacji JavaBean, który wykorzystywany może być przez narzędzia do wizualnego tworzenia aplikacji.
- Każde ziarno to serializowalny obiekt z bezargumentowym konstruktorem, do którego właściwości można dostać się przy pomocy metod dostępowych.
- JavaBeans nie powinny być mylone z ziarnami EJB (Enterprise JavaBeans), architekturą komponentową dla aplikacji rozproszonych, która mimo pewnych podobieństw realizuje inne zadania.

Cechy ziaren

- Aby klasa Javy stała się ziarnem powinna spełniać następujące konwencje:
 - klasa powinna być umieszczona w pakiecie;
 - klasa musi posiadać konstruktor domyślny lub bezparametrowy;
 - klasa powinna implementować interfejs `java.io.Serializable`;
 - wszystkie pola klasy muszą być prywatne;
 - klasa musi mieć metody dostępne do pól;
 - mogą być w klasie również metody nasłuchu i obsługi zdarzeń.

Właściwości

- Ziarna mają właściwości określające stan obiektu.
- Dostęp do właściwości zapewniają metody w klasie ziarna, zwane *akcesorami*:
 - akcesory odczytujące właściwości nazywane są *getterami*,
 - akcesory ustalające nowe wartości właściwości ziarna nazywane są *setterami*.
- Sposoby odczytywania i zmieniania stanu obiektu JavaBean:
 - standardowe wzorce deklaracji metod,
 - introspekcja realizowana za pomocą klasy Introspector.

Właściwości

- Typy właściwości:
 - właściwości proste (jedna wartość),
 - właściwości indeksowane (wiele wartości umieszczonych w tablicy).
- Getter dla właściwości prostej o nazwie *Prop* i typie *Type*:
`Type getProp () {...}`
`boolean isProp () {...}`
- Setter dla właściwości prostej o nazwie *Prop* i typie *Type*:
`void setProp (Type v) {...}`
- Getter dla właściwości indeksowanej o nazwie *Prop* i typie *Type*:
`Type getProp (int i) {...}`
`Type[] getProp () {...}`
- Setter dla właściwości indeksowanej o nazwie *Prop* i typie *Type*:
`void setProp (int i, Type v) {...}`
`void setProp (Type[] arr) {...}`

Właściwości

- Właściwość ziarna może być powiązana (ang. *bounded*):
 - o zmianie właściwości nieograniczonej mogą być powiadamiane inne ziarna
 - i mogą one reagować na tę zmianę.
- Właściwość ziarna może być ograniczona (ang. *constrained*):
 - o zmianie właściwości ograniczonej mogą być powiadamiane inne zainteresowane ziarna,
 - są one pytane o zgodę na tę zmianę i jeśli któreś z zainteresowanych ziaren zawetuje ją (nie da zgody) to zmiana nie dochodzi do skutku.

Nasłuch zmian

- Setter zarówno powiązanej jak i ograniczonej właściwości musi wygenerować zdarzenie `PropertyChangeEvent`.
- Ziarna mające właściwości powiązane powinny dostarczyć metody:

```
public void addPropertyChangeListener  
    (PropertyChangeListener) {...}  
public void removePropertyChangeListener  
    (PropertyChangeListener) {...}
```

a także metodę:

```
public PropertyChangeListener[]  
    getPropertyChangeListeners () {...}
```
- Ziarna mające właściwości ograniczone powinny dostarczyć metody:

```
public void addVetoableChangeListener  
    (VetoableChangeListener) {...}  
public void removeVetoableChangeListener  
    (VetoableChangeListener) {...}
```

a także metodę:

```
public VetoableChangeListener[]  
    getVetoableChangeListeners () {...}
```

Własne ziarna

Zdefiniowanie własnego ziarna wymaga zdefiniowania klasy, która:

- stosuje ogólnie przyjęte wzorce sygnatur metod oraz ewentualnie uzupełniona jest przez dodatkową klasę opisującą informacje o ziarnie (implementacja interfejsu `BeanInfo`);
- zapewnia serializację obiektów;
- posiada konstruktor bezparametrowy;
- uwzględnia pracę w środowisku wielowątkowym.

Nasłuch i wetowanie zmian

- Zmiana właściwości generuje zdarzenie `PropertyChangeEvent`.
- Komponenty zainteresowane śledzeniem zmian pewnej właściwości muszą implementować interfejs `PropertyChangeListener`.
- Komponenty, które mogą wetować zmiany pewnej właściwości muszą implementować interfejs `VetoableChangeListener`.

Nasłuch i wetowanie zmian

Obiekt zdarzenia typu `PropertyChangeEvent` możemy zapytać o:

- nazwę właściwości

```
String getPropertyName ()
```

- starą (przed zmianą) wartość właściwości

```
Object getOldValue ()
```

- nową (aktualną) wartość właściwości

```
Object getNewValue ()
```

Nasłuch i wetowanie zmian

- **Interfejs `PropertyChangeListener` ma jedną metodę:**

```
public void propertyChange  
    (PropertyChangeEvent ev);
```

- **Interfejs `VetoableChangeListener` ma jedną metodę:**

```
public void vetoableChange  
    (PropertyChangeEvent ev)  
    throws PropertyVetoException;
```

- **Wetowanie zmiany polega na zgłoszeniu wyjątku `PropertyVetoException`.**

Nasłuch zmian

- Słuchacze zmian właściwości powiązanych muszą zostać przyłączeni do źródła zdarzenia, czyli do ziarna.
- Słuchacza przyłączamy do ziarna za pomocą metody `addPropertyChangeListener(...)`.
- W ziarnie powinna też być zdefiniowana metoda odłączająca słuchacza od ziarna `removePropertyChangeListener(...)`.
- Każdy setter zmieniający właściwość powiązaną musi wygenerować zdarzenie `PropertyChangeEvent` i rozpropagować je wśród przyłączonych słuchaczy.

Nasłuch zmian

- Istnieje klasa narzędziowa `PropertyChangeSupport`, ułatwiająca przyłączanie i odłączanie słuchaczy oraz propagowanie zdarzeń wśród przyłączonych słuchaczy zmian określonej właściwości. Klasa ta posiada metody:
 - przyłączania i odłączania słuchaczy zmian
`addPropertyChangeListener (...)` i
`removePropertyChangeListener (...)`;
 - generujące i propagujące zdarzenia zmiany
`firePropertyChange (...)` i
`fireIndexedPropertyChange (...)`.

Wetowanie zmian

- Słuchacze zmian właściwości ograniczonych muszą zostać przyłączeni do źródła zdarzenia, czyli do ziarna.
- Słuchacza przyłączamy do ziarna za pomocą metody `addVetoableChangeListener(...)`.
- W ziarnie powinna też być zdefiniowana metoda odłączająca słuchacza od ziarna `removeVetoableChangeListener(...)`.
- Każdy setter zmieniający właściwość ograniczoną musi wygenerować zdarzenie `PropertyChangeEvent` i rozpropagować je wśród przyłączonych słuchaczy.

Wetowanie zmian

- Istnieje klasa narzędziowa

`VetoableChangeSupport`, ułatwiająca przyłączanie i odłączanie słuchaczy oraz propagowanie zdarzeń wśród przyłączonych słuchaczy zmian określonej właściwości. Klasa ta posiada metody:

- przyłączania i odłączania słuchaczy zmian
`addVetoableChangeListener(...)` i
`removeVetoableChangeListener(...)`;
- generujące i propagujące zdarzenia zmiany
`fireVetoableChange(...)`.

Klasa informacyjna ziarna

- Refleksja nie wystarcza – mogą istnieć metody `set/get`, które nie opisują właściwości ziarna.
- Bardziej uniwersalny mechanizm, to klasa implementująca `BeanInfo` – nazwa tej klasy powinna kończyć się przyrostkiem `BeanInfo` oraz należeć do tego samego pakietu co ziarno.
- Prosty rozwiązaniem na utworzenie klasy informacyjnej dla ziarna jest dziedziczenie po `SimpleBeanInfo`.

Klasa informacyjna ziarna

- Dla każdej udostępnionej właściwości ziarna tworzymy obiekt

PropertyDescriptor:

```
new PropertyDescriptor ("właściwość", Ziarno.class);
```

- Następnie implementujemy metodę getPropertyDescriptors ()
interfejsu BeanInfo:

```
class ZiarnoBeanInfo extends SimpleBeanInfo
```

```
{
```

```
    public PropertyDescriptor[]
```

```
    getPropertyDescriptors ()
```

```
    {
```

```
        return new PropertyDescriptor[]
```

```
        {
```

```
            new PropertyDescriptor
```

```
                ("właściwość1", Ziarno.class);
```

```
            new PropertyDescriptor
```

```
                ("właściwość2", Ziarno.class);
```

```
            ...
```

```
        };
```

```
    }
```

```
}
```

Edytory właściwości

- Jeśli właściwość ziarna jest liczbą lub napisem, to zostanie automatycznie umieszczona w oknie inspektora właściwości.
- Jeśli właściwość ziarna nie może być edytowana w polu tekstowym (daty, kolory itp.), należy dostarczyć własny edytor właściwości.
- Do określenia własnego edytora właściwości służy metoda `setPropertyEditorClass()`:

```
PropertyDescriptor desc =  
    new PropertyDescriptor(  
        "właściwość", Ziarno.class);  
desc.setPropertyEditorClass(  
    Edytor.class);
```

Edytory właściwości

- Edytor właściwości musi posiadać konstruktor domyślny oraz implementować interfejs `PropertyEditorSupport`.
- Dla każdego edytora właściwości wybieramy jeden z trzech sposobów wyświetlania i edycji właściwości:
 - jako łańcuch znaków (metody `getAsText` i `setAsText`);
 - jako pole wyboru (metody `getAsText`, `setAsText` i `getTags`);
 - graficznie poprzez rysunek (metody `isPaintable`, `paintValue`, `supportsCustomEditor` i `getCustomEditor`).

Literatura (JavaBeans)

- K.Barteczko: *Java – od podstaw do technologii. Tom 2, część B, rozdział 2: programowanie komponentowe z JavaBeans*. Wydawnictwo MIKOM, Warszawa 2004.
- C.S.Horstmann, G.Cornell: *Core Java – techniki zaawansowane. Wydanie 8. Rozdział 8: JavaBeans*. Wydawnictwo HELION, Gliwice 2009.
- JavaBeans(TM) Tutorial:
<http://download.oracle.com/javase/tutorial/javabeans/>