Programowanie współbieżne i rozproszone

WYKŁAD 9

dr inż. Krzysztof Pancerz

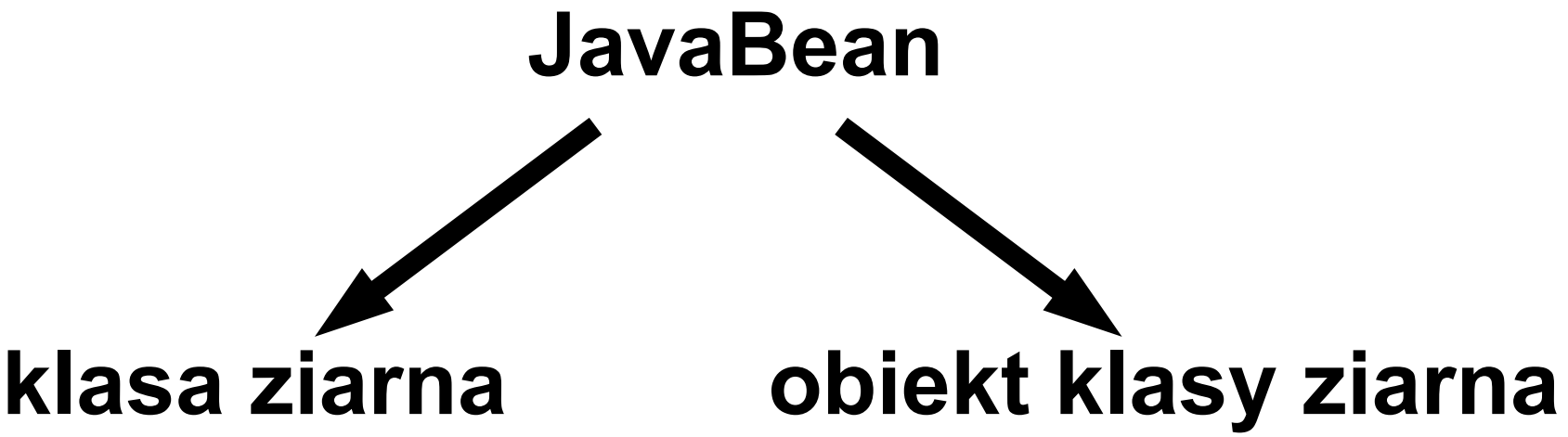
Komponenty

- Komponenty posiadają następujące cechy
  - są odpowiednio odizolowane od środowiska i innych komponentów
  - są wystarczająco samodzielne
  - są zaopatrzone w odpowiednią specyfikację, określającą jego wymagania i możliwości
  - obudowują swoją implementację i współpracują z otoczeniem za pośrednictwem dobrze określonego interfejsu

JavaBean

- *Bean* – ziarno
- JavaBean – programowy komponent "wielokrotnego użytku", którego właściwości i funkcjonalność mogą być odczytywane i/lub zmieniane uniwersalnymi środkami programistycznymi.
- Klasy-ziarna są typowymi klasami języka Java, które stosują odpowiedni interfejs programistyczny.
- Np. wszystkie komponenty pakietu Swing są ziarnami.

JavaBean (cd.)



Rozróżnienie wynika z kontekstu.

Właściwości i akcesory

- Ziarna posiadają właściwości (atrybuty).
- Dostęp do właściwości zapewniają metody klasy-ziarna nazywane akcesorsami.
- *getter* – akcesor pobierający właściwość
- *setter* – akcesor ustawiający właściwość
- Rodzaje właściwości:
  - proste (właściwości posiadające jedną wartość)
  - indeksowane (właściwości posiadające wiele wartości umieszczonych w tablicy)

Akcesory

- Akcesory dla właściwości prostej niebinarnej:
  - getter:

```
T getNNN( )
```

NNN – nazwa właściwości

T – typ właściwości

  - setter:

```
void setNNN(T)
```

NNN – nazwa właściwości

T – typ właściwości

Akcesory (cd.)

- Akcesory dla właściwości prostej binarnej:
  - getter:

```
boolean isNNN( )
```

NNN – nazwa właściwości

  - setter:

```
void setNNN(boolean)
```

NNN – nazwa właściwości

Akcesory (cd.)

- Akcesory dla właściwości indeksowanej:
  - getter (dla elementu):

```
T getNNN(int)
```

NNN – nazwa właściwości

T – typ elementów właściwości

  - getter (dla tablicy):

```
T[] getNNN()
```

NNN – nazwa właściwości

T – typ elementów właściwości

Akcesory (cd.)

- Akcesory dla właściwości indeksowanej:
  - setter (dla elementu):

```
void setNNN(int, T)
```

NNN – nazwa właściwości

T – typ elementów właściwości

  - setter (dla tablicy):

```
void setNNN(T[])
```

NNN – nazwa właściwości

T – typ elementów właściwości

Przykład ziarna

- JTextComponent
  - przykładowe akcesory:
    - String getText( )
    - void setText(String)
    - boolean isEditable( )
    - void setEditable(boolean)

Właściwości ziarna

- Właściwości związane (*bounded*)
  - o zmianie związanej właściwości ziarna mogą być informowane inne komponenty, które następnie mogą reagować na tą zmianę
- Właściwości ograniczone (*constrained*)
  - o zmianie ograniczonej właściwości ziarna mogą być informowane i pytane o zgodę inne komponenty
  - jeśli którykolwiek z poinformowanych komponentów nie wyrazi zgody na zmianę właściwości ograniczonej, zmiana ta nie dochodzi do skutku

Zdarzenia związane z akcesorami

- Akcesor ustawiający (*setter*) właściwość związaną lub ograniczoną ma obowiązek wygenerować zdarzenie klasy **PropertyChangeEvent**.
- Klasy-ziarna posiadające właściwości związane muszą dostarczać metody przyłączania słuchaczy zmian właściwości:

addPropertyChangeListener(PropertyChangeListener)

Zdarzenia związane z akcesorami (cd.)

- Klasy-ziarna posiadające właściwości ograniczone muszą dostarczać metody przyłączania słuchaczy:

addVetoableChangeListener(VetoableChangeListener)

Tworzenie klas-ziaren

- Klasa-ziarno musi spełniać następujące wymagania:
  - klasa stosuje ogólnie przyjęte wzorce sygnatur metod i/lub uzupełniona jest przez dodatkową specjalną klasę opisującą niestandardowe informacje o ziarnie (implementacja interfejsu **BeanInfo**)
  - klasa zapewnia serializację obiektów
  - klasa posiada konstruktor bezargumentowy
  - klasa uwzględnia działania w środowisku wielowątkowym (do obiektu-ziarna może równocześnie odwoływać się wiele wątków)

Nasłuch i wetowanie zmian właściwości

- Komponenty (obiekty), które mają śledzić zmiany właściwości związanej muszą implementować interfejs **PropertyChangeListener**, przez co stają się słuchaczami zmian właściwości.
- Komponenty (obiekty), które mają wetować zmiany właściwości związanej muszą implementować interfejs **VetoableChangeListener**, przez co stają się słuchaczami zmian właściwości i będą mogły wetować te zmiany.

Klasa PropertyChangeEvent

- Wybarne metody klasy **PropertyChangeEvent**:
  - **`String getPropertyName( )`** - zwraca nazwę zmienionej właściwości
  - **`Object getOldValue( )`** - zwraca starą wartość właściwości (przed zmianą)
  - **`Object getNewValue( )`** - zwraca nową wartość właściwości (po zmianie)

Interfejs PropertyChangeListener

- Interfejs **PropertyChangeListener** posiada jedną metodę:

```
public void propertyChange(PropertyChangeEvent)
```

- W metodzie tej można zareagować na zmianę danej właściwości.

Interfejs VetoableChangeListener

- Interfejs **VetoableChangeListener** posiada jedną metodę:

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

- W metodzie tej można zareagować na zmianę danej właściwości i ją ewentualnie zawetować.
- Gdy dana zmiana ma być zawetowana należy zgłosić wyjątek **PropertyVetoException**:

```
throw new PropertyVetoException(...)
```

Klasy generowania i propagacji zdarzeń zmian

- Klasa **PropertyChangeSupport**
  - konstruktor **PropertyChangeSupport(*ziarno*)**
  - metody **firePropertyChange** z różnymi argumentami
- Klasa **VetoableChangeSupport**
  - konstruktor **VetoableChangeSupport(*ziarno*)**
  - metody **fireVetoableChange** z różnymi argumentami

Refleksja

- Refleksja – możliwość podjęcia w trakcie wykonywania programu następujących działań:
  - uzyskanie pełnej informacji o charakterystykach klas (pola, metody, ich charakterystyki)
  - działanie na polach danego obiektu poprzez ich nazwy (a nie identyfikatory)
  - aktywowanie metod na rzecz danego obiektu poprzez ich nazwy i podanie argumentów

Refleksja (cd.)

- Refleksja pozwala m.in. na:
  - stwierdzenie jakie i z jakimi argumentami metody występują w danej klasie
  - dynamiczne wywoływanie metod (specyfikowanych w trakcie wykonywania programu)
  - dynamiczne uzyskiwanie i modyfikacje wartości pól obiektów

Introspekcja

- Introspekcja:
  - analiza komponentów za pomocą metod refleksji przy założeniu, że stosowane są pewne standardowe wzorce nazewnictwa umożliwiające określenie właściwości i ich typów, metod pobierania i ustawiania tych właściwości
  - analiza rodzajów zdarzeń, metod przyłączania słuchaczy i innych metod udostępnianych przez ziarna na zewnątrz, określanie uzewnętrznianych właściwości i funkcjonalności ziaren

Serializacja JavaBeans

- Obiekty-ziarna można serializować w postaci XML
- Zapisywaniem obiektów zajmuje się klasa **XMLEncoder**
- Odtwarzaniem obiektów zajmuje się klasa **XMLDecoder**