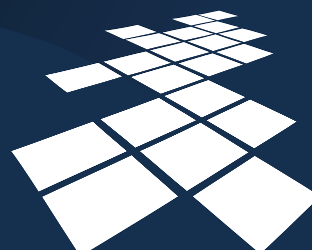
Logika biznesowa

wykład prowadzi
Mikołaj Morzy



Plan wykładu

- Komponenty JavaBean – wprowadzenie
- Komponenty JavaBean w JSP
- Wzorce projektowe
- JSTL – przykładowa biblioteka znaczników
  - znaczniki podstawowe
  - znaczniki formatujące
  - znaczniki XML
  - znaczniki SQL
- Tworzenie własnych znaczników

Celem wykładu jest przedstawienie metod tworzenia logiki biznesowej przy wykorzystaniu komponentów JavaBean i bibliotek znaczników JSP. Na początku zaprezentowana zostanie koncepcja komponentów JavaBean wraz z ich specyfikacją. Następnie omówione zostaną zagadnienia związane z wykorzystaniem komponentów JavaBean na stronach JSP. W szczególności omówione zostaną wzorce projektowe wykorzystujące komponenty JavaBean do komunikacji między poszczególnymi warstwami aplikacji internetowej. Druga część wykładu poświęcona będzie technologii bibliotek znaczników JSP – podstawowemu narzędziu służącemu do tworzenia dużych i złożonych aplikacji JSP. Koncepcja i implementacja mechanizmu bibliotek znaczników zostaną przedstawione na podstawie JSTL (ang. Java Standard Tag Library), standardowej biblioteki znaczników. Przykłady zilustrują sposób wykorzystania kilku rodzajów znaczników, m.in. znaczników podstawowych, formatujących, przetwarzających XML oraz obsługujących połączenie z bazą danych. Na koniec przedstawiona zostanie metoda tworzenia własnych znaczników JSP.

Co to jest komponent JavaBean?

- Napisany w języku Java komponent programowy wielokrotnego użytku, którym można manipulować wizualnie za pomocą odpowiedniego narzędzia (Sun)
- Niezależna od platformy sprzętowo-programowej obiektowa technologia tworzenia przenośnych komponentów wielokrotnego użytku
- Zastosowania:
  - graficzny interfejs użytkownika
  - komunikacja z bazą danych
  - aplikacje internetowe

Wiele kontrowersji i niejasności budzi samo pojęcie komponentu JavaBean. Zgodnie z definicją Sun Microsystems, twórcy koncepcji komponentów JavaBean, "komponentem JavaBean jest komponent programowy wielokrotnego użytku przeznaczony do tworzenia większych systemów. Własności komponentu JavaBean mogą być łatwo manipulowalne przy użyciu narzędzia graficznego". Początkowo więc komponenty JavaBean były pomyślane jako kontrolki oferujące określoną funkcjonalność za pomocą sprecyzowanego interfejsu. Komponenty JavaBean nie zdobyły wielkiej popularności jako składowe budulcowe większych systemów w sposób przewidziany przez twórców. Jednakże, szybko okazało się, że koncepcja jest przydatna i popularna w oderwaniu od wizualnych środowisk tworzenia oprogramowania. Dziś definiuje się JavaBean jako obiektowo zorientowaną technologię tworzenia przenośnych komponentów wielokrotnego użytku, w której poszczególne komponenty spełniają kilka prostych założeń (przedstawionych na następnym slajdzie). Komponenty JavaBean mogą być stosowane do tworzenia graficznego interfejsu użytkownika, ale przede wszystkim są wykorzystywane jako narzędzie do komunikacji z relacyjną bazą danych i jako podstawowy środek komunikacji między poszczególnymi warstwami aplikacji internetowej.

Aplikacje WWW

Kiedy klasa staje się komponentem JavaBean?

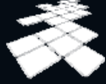
- Brak interfejsu, pakiet pomocniczy **`java.beans`**
- Klasa w pakiecie
- Konstruktor domyślny lub bezparametrowy
- Klasa implementuje interfejs **`java.io.Serializable`**
- Dla każdej cechy **`MyProperty`** metody dostępowe:
  - odczyt: **`getMyProperty()`**
  - odczyt (wartość logiczna): **`isMyProperty()`**
  - zapis: **`setMyProperty()`**
- Metody nasłuchu i obsługi zdarzeń

Logika biznesowa (4)

Aby klasa Java stała się komponentem JavaBean, nie musi implementować żadnego interfejsu, lecz powinna spełniać pewną konwencję dotyczącą nazewnictwa, konstruktorów, serializacji i metod dostępowych. Język Java definiuje pomocniczy pakiet java.beans umożliwiający tworzenie dużych, złożonych komponentów JavaBean, lecz klasy i interfejsy zawarte w tym pakiecie nie są wymagane do budowania komponentów JavaBean. Klasa Java, która funkcjonuje jako komponent JavaBean, powinna spełniać następujące konwencje:

- Klasa powinna (nie musi) być umieszczona w pakiecie
- Klasa musi posiadać konstruktor domyślny lub bezparametrowy
- Klasa powinna (nie musi) implementować interfejs java.io.Serializable w celu zapewniania trwałości komponentu
- Wszystkie składowe klasy (zwane dalej cechami) są prywatne, a dostęp do nich odbywa się za pomocą specjalnych metod dostępowych (ang. mutator methods) o ustalonych nazwach: odczyt cechy odbywa się za pomocą metody getNazwaCechy() (dla cech logicznych dopuszcza się nazwę isNazwaCechy()) a zapis cechy odbywa się za pomocą metody setNazwaCechy(). Przykładowo, jeśli komponent posiada cechy int age, boolean smoker, to komponent JavaBean powinien implementować metody dostępowe int getAge(), boolean isSmoker(), void setAge(int), void setSmoker(boolean).

Oryginalna specyfikacja technologii JavaBean zawiera również wymagania dotyczące implementacji metod nasłuchu zmian cech (ang. event listeners) i metod obsługi zdarzeń (ang. event handlers). Metody takie są implementowane poprzez odpowiednie interfejsy. Metody te nie są użyteczne w przypadku wykorzystania komponentów JavaBean w aplikacjach internetowych i tym samym wykraczają poza zakres omawianego materiału.



Przykład komponentu JavaBean

UserBean.java

```
package mm.beans;                                          (1)

public class UserBean implements java.io.Serializable {    (2)

    private String username;                               (3)
    private String email;

    public UserBean() {}                                   (4)

    public String getUsername() { return username; }       (5)

    public void setUsername(String username) {
        this.username = username;                          (6)
    }

    public String getEmail() { return email; }             (5)

    public void setEmail(String email) {
        this.email = email;                                (6)
    }
}
```


Slajd przedstawia przykładową klasę będącą komponentem JavaBean. Klasa mieści się w pakiecie (1) i implementuje interfejs java.io.Serializable (2). Dzięki implementacji interfejsu Serializable zawartość komponentu JavaBean może być łatwo utrwalana na dysku przy pomocy standardowego mechanizmu serializacji. Jeśli jakieś składowe klasy nie mają być utrwalane, to muszą być poprzedzone modyfikatorem transient. Klasa może być serializowana standardowo, ale można także zaproponować własną metodę serializacji przez implementację metod writeObject(java.io.ObjectOutputStream) i readObject(java.io.ObjectInputStream). Cechy komponentu JavaBean są prywatne i niedostępne na zewnątrz (3). Dostęp do cech odbywa się tylko za pomocą metod dostępu, tzw. getterów (5) i setterów (6). Klasa obowiązkowo musi posiadać konstruktor bezparametrowy (4) lub domyślny.

Znacznik <jsp:useBean>

useBean.jsp

```
<%@page contentType="text/html"%>
<html>
  <body>
    <jsp:useBean id="user" class="mm.beans.UserBean" scope="page"/>
      <% user.setUsername("Jan Kowalski");
         user.setEmail("Jan.Kowalski@poczta.pl"); %>

      Nazywam się <%= user.getUsername() %>.
  </body>
</html>
```

- Umożliwia wykorzystanie komponentu JavaBean w stronie JSP
  - `id`: unikalny identyfikator komponentu
  - `class`: kwalifikowana nazwa klasy komponentu
  - `scope`: zasięg widoczności komponentu

Znacznik <jsp:useBean> umożliwia wykorzystanie komponentu JavaBean w stronie JSP. Znacznik umożliwia pozbycie się dużej części skryptletów ze strony JSP, co zwiększa jej czytelność i prostotę. Znacznik wymaga podania unikalnego identyfikatora, pod którym komponent będzie widziany na stronie JSP, oraz nazwy klasy stanowiącej implementację komponentu i zasięgu widoczności komponentu. Nazwa klasy musi być w pełni kwalifikowana (tzn. musi obejmować także pełną nazwę pakietu). Komponent może być widoczny w ramach czterech zasięgów widoczności: bieżącej strony (page), bieżącego żądania (request), bieżącej sesji (session), lub globalnie (application). Znacznik <jsp:useBean> w pierwszej kolejności sprawdza, czy w podanym zasięgu widoczności występuje już komponent z danej klasy. Jeśli tak, to istniejący komponent zostaje ponownie wykorzystany, jeśli nie, tworzony jest nowy komponent. Właśnie ta cecha powoduje, że komponenty JavaBean można wykorzystać do "przenoszenia" danych między kolejnymi stronami JSP lub serwletami i dokumentami JSP, co ilustruje kolejny przykład.

Przykład użycia <jsp:useBean>

SimpleBeanServlet.java

```
ServletContext ctx = this.getServletContext();
RequestDispatcher dispatcher =
   ctx.getRequestDispatcher("/SimpleBeanPage.jsp");

UserBean user = new UserBean();
user.setUsername("James Bond");
user.setEmail("bond007@hotmail.com");
request.setAttribute("user",user);
dispatcher.forward(request,response);
```

1 2 3 4

SimpleBeanPage.jsp

```
<%@page contentType="text/html"%>
<html>
  <body>
    <jsp:useBean id="user"
         class="mm.beans.UserBean" scope="request"/>
    My name is <%= user.getUsername() %>
    and my email is <%= user.getEmail() %>.
  </body>
</html>
```


Pierwszy listing przedstawia kod serwletu SimpleBeanServlet.java. Serwlet pobiera zarządcę żądań i wiąże zarządcę ze stroną SimpleBeanPage.jsp (1). Następnie serwlet tworzy nową instancję komponentu JavaBean (2) i wypełnia komponent danymi dotyczącymi nazwy użytkownika i adresu email (3). W kolejnym kroku serwlet umieszcza komponent JavaBean w zasięgu widoczności żądania i przekierowuje żądanie do strony SimpleBeanPage.jsp (4). Strona JSP wykorzystuje znacznik <jsp:useBean/> do znalezienia komponentu o podanej nazwie. Ponieważ w podanym zakresie widoczności (request) znajduje się komponent o podanej nazwie i klasie, zostaje on załadowany i wykorzystany do wyświetlenia na stronie. Jeśli serwlet nie umieściłby komponentu w zasięgu widoczności request, to znacznik <jsp:useBean> spowodowałby utworzenie nowej instancji komponentu JavaBean. Wówczas, rzecz jasna, odwołania do metod getUsername() i getEmail() zwróciłyby wartość null.

Znaczniki <jsp:getProperty> i <jsp:setProperty>

- Umożliwiają łatwe odwołania do metod dostępowych komponentu JavaBean z poziomu strony JSP
- Eliminują skryptlety
- Znacznik <jsp:setProperty>
  - ustawia podaną wartość jako cechę komponentu
  - ustawia wartość parametru jako cechę komponentu
  - wczytuje wszystkie parametry do komponentu

Oprócz znacznika <jsp:useBean> JSP oferuje dwa dodatkowe znaczniki umożliwiające dostęp do cech komponentu JavaBean i manipulowanie wartościami cech. Zastosowanie obu znaczników praktycznie eliminuje konieczność wykorzystywania skryptletów, co prowadzi do czystszego i prostszego kodu JSP. Znacznik <jsp:getProperty> powoduje odczytanie wartości podanej cechy w komponencie JSP, automatyczną konwersję zwróconej wartości do łańcucha znaków, i wyświetlenie wartości wynikowej na stronie JSP. Znacznik <jsp:setProperty> powoduje ustawienie wartości pewnej cechy. Nowa wartość może być podana bezpośrednio w kodzie lub może zostać odczytana z parametru HTTP, z jakim została wywołana strona JSP. Dla komponentów funkcjonujących jako obiektowa reprezentacja zbioru wartości odczytywanych z formularza HTML istnieje także wygodna metoda jednoczesnego pobrania wartości wszystkich parametrów i przypisania ich do właściwych cech komponentu JavaBean. Na następnym slajdzie przedstawiono przykład wykorzystania obu znaczników.

Aplikacje WWW

Przykład wykorzystania znaczników <jsp:getProperty> i <jsp:setProperty>

```
<%@page contentType="text/html"%>
<html>
  <body>
    <jsp:useBean id="user"
     class="mm.beans.FormBean" scope="request"/>

    <jsp:setProperty name="user" property="*"/>

    <jsp:setProperty name="user" property="email"
        value="bond007@hotmail.com"/>

  My name is
    <jsp:getProperty name="user" property="name"/>
  and I'm
    <jsp:getProperty name="user" property="age">.
  </body>
</html>
```

Logika biznesowa (9)

Slajd przedstawia przykładową stronę JSP wykorzystującą znaczniki <jsp:getProperty> i <jsp:setProperty> do manipulacji zawartością komponentu JavaBean. W pierwszym kroku następuje odszukanie komponentu JavaBean o podanej nazwie w określonym zakresie widzialności (1). Następnie, za pomocą znaczników <jsp:setProperty> następuje ustawienie wartości cech komponentu (2). Pierwszy znacznik powoduje odczytanie i dopasowanie parametrów HTTP do cech komponentu. Drugi znacznik przedstawia sposób ustawienia wartości cechy komponentu na podstawie znanej wartości. Wreszcie, wartości cech komponentu są wyświetlane na stronie JSP dzięki użyciu znaczników <jsp:getProperty> (3).

Aplikacje WWW

Inicjalizacja komponentu JavaBean

- Tradycyjnie komponent JavaBean inicjalizowany przez serwlet i przekazywany do strony JSP
- Inicjalizacja przez ciało znacznika **<jsp:useBean>**

```
<jsp:useBean id="user" scope="request"
     class="mm.beans.FormBean">
  <jsp:setProperty name="user" property="name" value="007"/>
  <jsp:setProperty name="user" property="age"  value="40"/>
</jsp:useBean>

My name is <jsp:getProperty name="user" property="name"/>
and I'm <jsp:getProperty name="user" property="age"/> old.
```

Logika biznesowa (10)

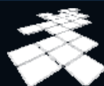
Najczęściej za inicjalizację komponentów JavaBean i ich osadzenie we właściwym zakresie widoczności odpowiedzialny jest serwlet. Czasem jednak tworzone są niewielkie aplikacje internetowe składające się z kilku powiązanych ze sobą stron JSP, w których użycie serwletu jest niepotrzebne. W takim przypadku inicjalizacja wartości cech komponentu JavaBean może nastąpić bezpośrednio na stronie JSP. Do inicjalizacji komponentu służy ciało znacznika <jsp:useBean>, które jest wykonywane tylko jeden raz, w momencie tworzenia komponentu. Jeśli komponent jest przekazywany między różnymi stronami JSP (np. poprzez zasięg widoczności request lub session), tylko pierwsza strona odwołująca się do komponentu faktycznie uruchamia ciało znacznika <jsp:useBean>.

Po co komponenty JavaBean?

- Zalety wykorzystania komponentów JavaBean
  - prosta i czytelna składnia
  - eliminacja skryptletów
  - separacja logiki biznesowej i warstwy prezentacji
  - wielokrotne wykorzystanie tej samej logiki biznesowej
  - możliwość podziału zespołu projektowego

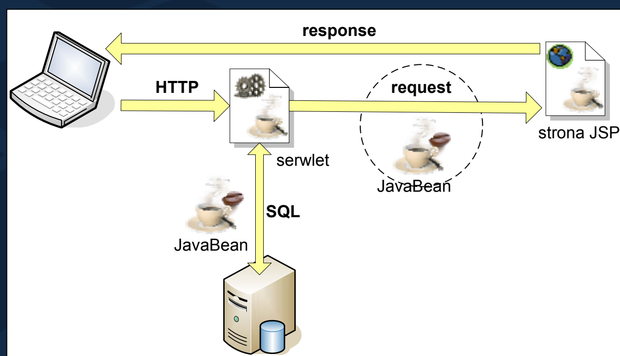
Komponenty JavaBean nie stanowią nowej jakości w programowaniu stron JSP, ale ich konsekwentne wykorzystanie niesie ze sobą wiele korzyści. Stosowanie znaczników <jsp:useBean> <jsp:getProperty> i <jsp:setProperty> wprowadza czystą i czytelną składnię oraz praktycznie eliminuje konieczność stosowania skryptletów na stronach JSP. W konsekwencji strony JSP stają się prostsze i łatwiejsze w utrzymaniu. Istotną zaletą wykorzystania komponentów JavaBean jest możliwość pełnej separacji implementacji logiki biznesowej (komponent JavaBean) i warstwy prezentacji, czyli generacji wynikowego kodu HTML (strona JSP). Taka separacja jest nie tylko wymogiem projektowym, ale umożliwia wielokrotne wykorzystanie tego samego kodu implementacji logiki biznesowej w wielu różnych miejscach, dzięki czemu można zapewnić identyczną funkcjonalność różnym modułom aplikacji internetowej. Dodatkowo, separacja logiki biznesowej i warstwy prezentacji umożliwia podział zespołu programistycznego na zespół implementujący komponenty JavaBean (osoby programujące w języku Java) i zespół generujący wynikowe dokumenty HTML (osoby programujące w języku HTML i zajmujące się grafiką).

Aplikacje WWW



Wzorce projektowe i JavaBeans

- Zasady
  - komponenty nieświadome środowiska
  - komponenty do logiki biznesowej i komunikacji
  - eliminacja skryptletów z dokumentów JSP



Logika biznesowa (12)

Efektywne korzystanie z komponentów JavaBean wymaga stosowania kilku wzorców i zasad projektowych. Komponenty JavaBean powinny być całkowicie nieświadome środowiska, w którym działają. Ten sam komponent JavaBean musi mieć możliwość działania zarówno w środowisku serwera aplikacji jako składowa aplikacji internetowej, jak i w tradycyjnej graficznej aplikacji użytkownika końcowego. Stąd, bardzo złą praktyką programistyczną jest uzależnianie działania komponentu JavaBean od specyficznych interfejsów środowiskowych. Przykładowo, niedopuszczalne jest przekazywanie do komponentu obiektów typu javax.servlet.http.HttpServletRequest, ponieważ ściśle wiąże to komponent JavaBean ze środowiskiem serwera aplikacji. Komponenty JavaBean służą do dwóch niezależnych celów: z jednej strony wykonują logikę biznesową (np. nawiązują połączenie z bazą danych i odczytują dane), z drugiej strony służą do komunikacji między warstwami aplikacji internetowej (np. przenoszą dane z serwletu do strony JSP). Te dwie funkcjonalności komponentów JavaBean nie powinny być ze sobą mieszane w celu zapewnienia przejrzystości i czytelności kodu. Wreszcie, konsekwentne wykorzystanie komponentów JavaBean powinno prowadzić do całkowitej eliminacji skryptletów ze stron JSP.

Na rysunku przedstawiono szkielet aplikacji internetowej zbudowanej wg powyższych zaleceń. Żądanie HTTP klienta jest kierowane do serwletu, który tworzy komponent JavaBean wykonujący logikę biznesową polegającą na połączeniu z serwerem bazy danych i weryfikacji nazwy i hasła użytkownika. Następnie, serwlet tworzy kolejny komponent JavaBean, wypełnia go potrzebnymi danymi (być może także danymi odczytanymi z bazy danych), umieszcza komponent w zasięgu widoczności request i przekazuje sterowanie do strony JSP. Strona JSP pobiera komponent za pomocą znacznika <jsp:useBean> i generuje wynikowy kod HTML przekazywany jako odpowiedź klientowi HTTP.

Przykład zaawansowanej aplikacji (1/5)

form.html

```
<html>
  <body>
    <form action="DBLoginServlet">
     userid:   <input name="userid" type="text"/><br/>
     password: <input name="password" type="password"/><br/>
     <input type="submit" value="login"/>
    </form>
  </body>
</html>
```

```
SQL> select * from users;

USERID     PASSWORD     USERNAME
---------- ------------ ---------------
jkowal     mypassword   Jan Kowalski
```


Kolejne slajdy przedstawiają przykład zaawansowanej aplikacji wykorzystującej serwlety, komponenty JavaBean i strony JSP do implementacji autoryzacji dostępu do strony na podstawie zawartości bazy danych. Plik form.html zawiera prosty formularz HTML umożliwiający wprowadzenie identyfikatora i hasła użytkownika. W bazie danych umieszczono tabelę users zawierającą informacje o użytkownikach i ich hasłach. Parametry z formularza przekazywane są do serwletu DBLoginServlet przedstawionego na następnym slajdzie.

Przykład zaawansowanej aplikacji (2/5)

DBLoginServlet.java

```
DBBean loginBean = new DBBean();
String userid = request.getParameter("userid");
String password = request.getParameter("password");
String username = null;

try {
  username = loginBean.getUsername(userid,password);
} catch (SQLException ex) { ex.printStackTrace(); }

UserBean userBean = new UserBean();
userBean.setUsername(username);
request.setAttribute("userbean",userBean);

ServletContext ctx = this.getServletContext();
RequestDispatcher dispatcher =
  ctx.getRequestDispatcher("/DBLogin.jsp");
dispatcher.forward(request,response);
```


Slajd przedstawia fragment metody doGet() serwletu DBLoginServlet. W pierwszym kroku tworzona jest nowa instancja komponentu DBBean służącego do komunikacji z bazą danych (1). Komponent ten zaszywa całą logikę nawiązania połączenia z bazą danych. Do połączenia z bazą danych potrzebne są informacje o identyfikatorze i haśle użytkownika, odczytywane z obiektu request. W kroku (2) następuje wywołanie metody getUsername() definiowanej w komponencie DBBean. Metoda ta łączy się z bazą danych i odczytuje rzeczywistą nazwę użytkownika. W kroku (3) tworzona jest instancja komponentu UserBean służącego do komunikacji i przenoszenia danych między poszczególnymi warstwami aplikacji internetowej. Nazwa użytkownika, odczytana z bazy danych, zostaje zapisana jako cecha komponentu, a sam komponent zostaje umieszczony w zakresie widoczności request. Ostatnią czynnością wykonywaną przez serwlet jest przekazanie sterowania do strony DBLogin.jsp (4).

Przykład zaawansowanej aplikacji (3/5)

DBBean.java

```
public String getUsername(String userid, String password)
       throws SQLException {

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@host:port:sid","user","pass");              1

  PreparedStatement stmt = conn.prepareStatement(
    "SELECT username FROM users WHERE userid = ? AND password = ?");
  stmt.setString(1,userid);
  stmt.setString(2,password);                                2
  ResultSet rset = stmt.executeQuery();
  String result = null;
  while (rset.next())                   3
    result = rset.getString("username");
  return result;
  }
```


Slajd przedstawia fragment kodu komponentu DBBean.java odpowiedzialny za implementację logiki biznesowej (w tym przypadku za połączenie z bazą danych). W kroku (1) następuje załadowanie sterownika JDBC i nawiązanie połączenia z bazą danych. W kroku (2) zapytanie zostaje przekazane do bazy danych. Należy zwrócić szczególną uwagę na zastosowanie późnego wiązania zmiennych, jest to kluczowa sprawa w przypadku wielodostępnych aplikacji internetowych. W kroku (3) następuje pobranie i zwrócenie wyniku zapytania.

Przykład zaawansowanej aplikacji (4/5)

UserBean.java

```
package mm.beans;

public class UserBean implements java.io.Serializable {

    private String username;

    public UserBean() {
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
}
```


Slajd przedstawia kod komponentu UserBean.java odpowiedzialnego za komunikację i przenoszenie danych między warstwą logiki biznesowej (reprezentowanej przez serwlet DBLoginServlet i komponent DBBean) i warstwą prezentacji implementowaną w postaci stron JSP. Jest to uproszczona wersja komponentu przedstawianego na jednym z wcześniejszych slajdów. Komponent posiada jedną cechę username, domyślny konstruktor i metody dostępowe getUsername() i setUsername().

Aplikacje WWW

Przykład zaawansowanej aplikacji (5/5)

DBLogin.jsp

```
<%@page contentType="text/html"%>
<html>
  <body>
    <jsp:useBean id="userbean" scope="request"
         class="mm.beans.UserBean"/>

    Welcome,
      <jsp:getProperty name="userbean" property="username"/>.
  </body>
</html>
```

http://localhost:8084/Logika__biznesowa/DBLoginServlet?userid=jkowal&password=mypassword

Welcome, Jan Kowalski.

userid: jkowal
password: ••••••••••
login

Logika biznesowa (17)

Ostatni slajd przedstawia końcową stronę JSP i ostateczny wynik aplikacji. Strona JSP odczytuje komponent UserBean z zasięgu widoczności request i zakłada, że warstwa logiki biznesowej przygotowała komponent poprawnie, tzn. że komponent zawiera odczytane z bazy danych informacje. Osoba tworząca stronę JSP nie musi pisać ani jednej linijki kodu, lecz porusza się tylko w obrębie statycznego języka HTML wzbogaconego o dane odczytywane z komponentu. Poniżej zrzuty ekranu pokazujące ostateczny wygląd i zachowanie aplikacji. Dla uproszczenia przykładu pominięto istotną kwestię obsługi błędów (np. podanie nieprawidłowego hasła lub nieistniejącego identyfikatora użytkownika).

Co to jest biblioteka znaczników?

- Osiem znaczników (akcji) standardowych
  – dostęp do obiektów
  – modyfikacja wyglądu strony
- Od wersji JSP 1.1 możliwość definiowania znaczników przez użytkownika
- Biblioteka znaczników to zbiór znaczników zdefiniowanych przez użytkownika
  – przetwarzanie formularzy
  – komunikacja z bazą danych
  – kontrola przepływu logiki
  – generowanie wykresów

Specyfikacja JSP definiuje osiem standardowych znaczników akcji, przedstawionych w trakcie wcześniejszego wykładu. Znaczniki te umożliwiają dostęp do obiektów, wykorzystanie komponentów JavaBean, modyfikację wyglądu dokumentu wynikowego, usunięcie skryptletów ze strony JSP. Począwszy od wersji 1.1 specyfikacja JSP umożliwia użytkownikom tworzenie własnych znaczników (ang. custom tags). Dla każdego zdefiniowanego przez użytkownika znacznika należy także przygotować procedurę obsługi znacznika, tzn. kod który będzie uruchamiany w momencie napotkania danego znacznika. Znaczniki definiowane przez użytkownika posiadają podobną funkcjonalność do komponentów JavaBean, jednak w porównaniu z komponentami umożliwiają bardziej modularną i czytelną konstrukcję stron JSP. Należy też wspomnieć o podstawowej zalecie komponentów JavaBean w porównaniu ze znacznikami JSP: implementacja logiki biznesowej w postaci komponentu JavaBean nie wiąże implementacji jednoznacznie ze środowiskiem aplikacji internetowych, jak ma to miejsce w przypadku znaczników definiowanych przez użytkownika.

Zbiór znaczników o podobnej funkcjonalności może zostać skonsolidowany do postaci biblioteki znaczników (ang. tag library). Istnieje wiele komercyjnych i publicznych bibliotek znaczników oferujących bardzo różnorodną funkcjonalność. Biblioteki znaczników umożliwiają, między innymi, łatwe przetwarzanie zawartości formularzy HTML, komunikację z relacyjnymi bazami danych, konstrukty sterujące przepływem logiki, dynamiczne generowanie wykresów i obrazów, zarządzanie uprawnieniami użytkowników, obsługę poczty elektronicznej, logowanie informacji, i bardzo wiele innych.

Znaczniki definiowane przez użytkownika

- Cechy znaczników definiowanych przez użytkownika
  - parametry przekazywane przez atrybuty
  - dostęp do obiektów na stronie JSP
  - modyfikacja dokumentu wynikowego
  - komunikacja między znacznikami
  - zagnieżdżanie znaczników
- Deklaracja wykorzystania biblioteki znaczników
  - `*.tld` : opis biblioteki znaczników

```
<%@ taglib uri="/WEB-INF/calendar.tld" prefix="calendar"/>
```


Znaczniki definiowane przez użytkownika, podobnie jak komponenty JavaBean, umożliwiają pełną separację warstwy implementacji logiki biznesowej i warstwy generowania widoku użytkownika oraz zapewniają przenaszalność funkcjonalności między różnymi stronami JSP. Znaczniki definiowane przez użytkownika mogą być dowolnie parametryzowane, wartości parametrów są przekazywane do znacznika za pomocą atrybutów. Procedury obsługi znaczników odpowiedzialne za faktyczną implementację logiki reprezentowanej przez znacznik mają pełny dostęp do wszystkich obiektów umieszczonych na danej stronie JSP. Znaczniki mają możliwość modyfikowania dokumentu wynikowego. Znaczniki umieszczone na tej samej stronie JSP mogą się ze sobą komunikować za pomocą komponentów JavaBean. Wreszcie, znaczniki definiowane przez użytkownika mogą być dowolnie zagnieżdżone, tworząc złożone struktury zależności między znacznikami. Z każdą biblioteką znaczników związany jest plik *.tld (ang. tag library descriptor) zawierający odwzorowanie poszczególnych znaczników na właściwe procedury obsługi znaczników. Użycie biblioteki znaczników na stronie JSP wymaga uprzedniego zadeklarowania chęci wykorzystania danej biblioteki. Konieczne jest wskazanie lokalizacji pliku z opisem biblioteki oraz podanie prefiksu wykorzystywanego dla znaczników z danej biblioteki.

Standardowa biblioteka JSTL

- JSTL (ang. JavaServer Pages Standard Tag Library)
  - specyfikacja rozwijana przez Sun: JSR-52
  - aktualna wersja: 1.2 (maj 2006)
  - podstawowa implementacja: Jakarta Taglibs
- Kategorie znaczników JSTL
  - ogólnego przeznaczenia, warunkowe i iteracyjne
  - formatowanie tekstu i internacjonalizacja
  - manipulacja dokumentami XML
  - dostęp do baz danych

Najczęściej wykorzystywaną biblioteką znaczników jest biblioteka JSTL (ang. JavaServer Pages Standard Tag Library). Jest to biblioteka, której specyfikację rozwija, w ramach Java Community Process JSR-52, firma Sun Microsystems. Aktualna wersja biblioteki to 1.2 (ostateczna wersja specyfikacji JSTL 1.2 została opublikowana w maju 2006 roku). Istnieje kilka implementacji specyfikacji, najpopularniejszą i najczęściej stosowaną jest bez wątpienia implementacja Apache Jakarta Taglibs. W ramach biblioteki JSTL można wyróżnić kilka podstawowych kategorii znaczników definiowanych przez użytkownika. Istnieją znaczniki umożliwiające implementację pętli iteracyjnych i instrukcji warunkowych, znaczniki do formatowania łańcuchów znaków, znaczniki służące do manipulacji zawartością dokumentów XML, czy wreszcie znaczniki umożliwiające wykorzystanie bazy danych.

Znaczniki podstawowe

- <@% taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
- Znaczniki ogólnego przeznaczenia
  - `<c:out> <c:set> <c:remove> <c:catch>`
- Znaczniki do iteracji
  - `<c:forEach> <c:forTokens>`
- Znaczniki wyboru warunkowego
  - `<c:if> <c:choose> <c:when> <c:otherwise>`
- Znaczniki do manipulacji adresami URL
  - `<c:url> <c:param> <c:redirect>`

Do pierwszej kategorii znaczników JSTL zaliczamy tzw. znaczniki podstawowe (ang. core tags) podzielone na podgrupy tematyczne.

Znaczniki ogólnego przeznaczenia:

<c:out>: ewaluuje wyrażenie i wypisuje wynik ewaluacji na stronie JSP

<c:set>: ustawia wartość cechy komponentu JavaBean lub wartość zmiennej dostępnej dla innych znaczników na stronie

<c:remove>: usuwa zmienną z podanego zakresu widoczności, np. podczas czyszczenia sesji

<c:catch>: przechwytuje wyjątek i zapisuje wyjątek do zmiennej

Znaczniki umożliwiające iterację

<c:forEach>: iteracja określoną liczbę razy lub iteracja po wszystkich elementach kolekcji (tablica, java.util.Collection, java.util.Iterator, java.util.Enumeration, java.util.Map)

<c:forTokens>: analizuje łańcuch znaków i dzieli go na fragmenty zgodnie z podanym separatorem, następnie iteruje po wszystkich elementach powstałych z dzielonego łańcucha

Znaczniki wyboru warunkowego

<c:if>: wykonuje fragment strony JSP jeśli spełniony jest warunek

<c:choose><c:when><c:otherwise>: tworzą blok wyboru wielokrotnego

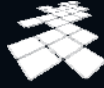
Znaczniki do manipulacji adresami URL

<c:url>: koduje adres URL wraz z informacjami o sesji użytkownika i parametrami HTTP

<c:param>: wprowadza parametr HTTP (używany w ciele znacznika <c:url>)

<c:redirect>: przekierowuje sterowanie do podanego adresu URL

Aplikacje WWW



Przykład wykorzystania znaczników 1/3

```
<%@page contentType="text/html"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
  <body>
    <form>
      Podaj <b>n</b>: <input name="n" type="text"/> <br/>
      angielski <input type="radio" name="language" value="en"/>
      polski    <input type="radio" name="language" value="pl"/>
      <br/> <input type="submit" value="Oblicz n!"/>
    </form>

    <c:if test='${! empty param.n}'>
      <c:catch var="e">

        <c:set var="n" scope="page" value="${param.n}"/>
        <c:set var="silnia" scope="session" value="1"/>

        <c:forEach var="i" begin="1" end="${n}">
          <c:set var="silnia" value="${silnia*i}" scope="session"/>
        </c:forEach>
```

Logika biznesowa (22)

Kolejne trzy slajdy przedstawiają przykład prostej aplikacji wykorzystującej tylko i wyłącznie znaczniki podstawowe JSTL. W kroku (1) tworzony jest formularz HTML umożliwiający użytkownikowi wprowadzenie parametru liczbowego oraz wyboru języka, w jakim zostanie wyświetlony wynik. Warto zwrócić uwagę na to, że znacznik <form> nie posiada atrybutu action. W takim przypadku w momencie wysłania formularza bieżąca strona JSP zostanie załadowana powtórnie z ustawionymi wartościami parametrów wprowadzonych przez użytkownika. Ponieważ podczas pierwszego wyświetlenie strony JSP wartości parametrów nie są jeszcze ustalone, w kroku (2) wykorzystujemy znacznik <c:if> do sprawdzenia, czy wartość parametru n jest ustalona. Pozostała część strony jest przetwarzana tylko podczas drugiego i kolejnych wyświetleń strony JSP. Jednym z błędów, jakie mogą wystąpić, jest wpisanie przez użytkownika wartości parametru n która nie jest liczbą. Aby obsłużyć takie zdarzenie cały dalszy blok kodu jest ujęty w znaczniku <c:catch> umożliwiającym przechwycenie błędu konwersji (3). W kroku (4) deklarujemy dwie zmienne za pomocą znacznika <c:set>: zmienna n o zasięgu widoczności bieżącej strony służy do przechowywania wartości parametru i funkcjonuje jak zmienna lokalna. Zmienna silnia o zasięgu sesji posłuży nam do przekazania wartości do innej strony JSP. Krok (5) to wykorzystanie znacznika <c:forEach> w celu iteracyjnego wyliczenia wartości silnia(n). Kod programu jest kontynuowany na kolejnym slajdzie.

Przykład wykorzystania znaczników 2/3

```
        ...
        <c:url value="display.jsp" var="display">
          <c:param name="language" value="${param.language}"/>
        </c:url>
        <c:redirect url="${display}"/>
      </c:catch>

      <c:if test="${! empty e}">
        <c:out value="${e}"/>
      </c:if>
    </c:if>

  </body>
</html>
```

6

7

http://localhost:8084/Logika__biznesowa/JSTLExample.jsp?n=abcdefgh&language=pl - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Adres http://localhost:8084/Logika__biznesowa/JSTLExample.jsp?n=abcdefgh&language=pl

Podaj n:

angielski polski

Oblicz n!

javax.servlet.jsp.el.ELException: An exception occured trying to convert String "abcdefgh" to type "java.lang.Integer"

Krok (6) to wykorzystanie znacznika <c:url> do skonstruowania i zakodowania adresu URL, pod który nastąpi przekierowanie. Dodatkowo, zagnieżdżony znacznik <c:param> powoduje dodanie do adresu URL parametru zawierającego informację o wybranym przez użytkownika języku wyświetlania wyniku. Po zamknięciu bloku <c:catch> w kroku (7) sprawdzamy, czy w trakcie wykonywania strony JSP wystąpił jakiś błąd (wystąpienie błędu jest sygnalizowane tym, że zmienna e związana ze znacznikiem <c:catch> zawiera komunikat z informacją o błędzie). W przypadku wystąpienia błędu komunikat o błędzie jest wyświetlany za pomocą znacznika <c:out>. Zrzut ekranu przedstawia zachowanie się aplikacji w przypadku podania łańcucha znaków "abcdefgh" jako wartości parametru n.

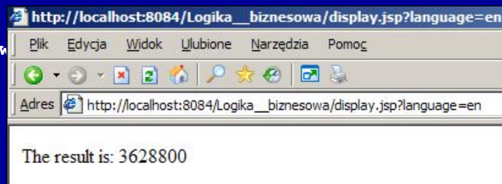
Przykład wykorzystania znaczników 3/3

display.jsp

```
<%@page contentType="text/html"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
  <body>

    <c:choose>  (1)
      <c:when test="${param.language == 'en'}">
        The result is: <c:out value="${silnia}"/>  (2)
      </c:when>
      <c:when test="${param.language == 'pl'}">
        Wynik to: <c:out value="${silnia}"/>
      </c:when>
      <c:otherwise>
        Nie wiem, jaki jest w
      </c:otherwise>
    </c:choose>

  </body>
</html>
```



Powyższy slajd przedstawia kod umieszczony na stronie display.jsp, do której następuje przekierowanie. W kroku (1) znacznik <c:choose> otwiera instrukcję wyboru wielokrotnego. Poszczególne opcje są dostępne wewnątrz znaczników <c:when> (w tym przypadku wybór uzależniony jest od wartości parametru language). Warto zwrócić szczególną uwagę na punkt (2), wyświetlana zmienna silnia nie jest zadeklarowana w kodzie strony display.jsp. W rzeczywistości jest to zmienna o zasięgu widoczności sesji zadeklarowana i zainicjalizowana przez stronę z której nastąpiło przekierowanie. Zmienna ta zatem działa jak komponent JavaBean przenoszący informację między poszczególnymi stronami składającymi się na aplikację. Zrzut ekranu przedstawia wynik działania aplikacji przy wyborze języka angielskiego jako języka wyświetlania wyniku.

Aplikacje WWW

Znaczniki formatujące

- <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

- Znaczniki obsługujące pakiety językowe
  - **`<fmt:setBundle>` `<fmt:setLocale>`**
- Znaczniki obsługujące strefę czasową
  - **`<fmt:setTimeZone>`**
- Znaczniki formatujące daty i liczby
  - **`<fmt:formatNumber>` `<fmt:formatDate>`**
- Inne znaczniki
  - **`<fmt:message>` `<fmt:param>`**

Logika biznesowa (25)

Drugą kategorię znaczników JSTL stanowią znaczniki formatujące. Można je podzielić na następujące podkategorie.

Znaczniki obsługujące pakiety

<fmt:setBundle>: wskazuje na pakiet plików z ustawieniami językowymi który będzie domyślnie wykorzystywany przez znaczniki formatujące

<fmt:bundle>: wskazuje na pakiet plików z ustawieniami językowymi który będzie wykorzystywany przez zagnieżdżone znaczniki formatujące

<fmt:setLocale>: powoduje ustawienie określonej lokalizacji

Znaczniki obsługujące strefę czasową

<fmt:setTimeZone>: powoduje ustawienie określonej strefy czasowej

Znaczniki obsługujące daty i liczby

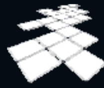
<fmt:formatNumber>: powoduje sformatowanie podanej liczby, możliwe zmiany symboli i separatorów, zaokrąglanie, wyświetlanie w postaci waluty, itp.

<fmt:formatDate>: powoduje sformatowanie podanej daty do jednego z wielu dostępnych formatów (krótki, średni, pełny)

Inne znaczniki

<fmt:message>: wyświetla wartość przypisaną do podanego klucza w pliku z ustawieniami językowymi

<fmt:param>: umożliwia parametryzację komunikatów wyświetlanych za pomocą znacznika <fmt:message>



Przykład wykorzystania znaczników 1/2

```
<%@page contentType="text/html"%>

<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
  <body>
    <fmt:setBundle basename="messages"/>          (1)

    <h1><fmt:message key="title"/></h1>

    <c:set var="salary" value="1200"/>            (2)
    <jsp:useBean id="now" class="java.util.Date"/>

(3) <fmt:message key="salary_label"/>
      <fmt:formatNumber value="${salary}" type="currency"/> br/>
    <fmt:message key="date_label"/>
(4)   <fmt:formatDate value="${now}" dateStyle="long"/>

  </body>
</html>
```


Kolejne dwa slajdy przedstawiają przykład aplikacji wykorzystującej znaczniki formatujące do internacjonalizacji aplikacji internetowej. W kroku (1) następuje załadowanie pakietu plików z ustawieniami językowymi (postać pakietu plików jest pokazana na następnym slajdzie). Znacznik <fmt:message> jest wykorzystany do wyświetlenia komunikatu związanego z kluczem "title" (komunikat, a właściwie różne wersje językowe tego komunikatu, mieszczą się w pakiecie plików z ustawieniami językowymi). W kroku (2) tworzymy zmienną liczbową zawierającą pensję i zmienną zawierającą bieżącą datę. Krok (3) to przykład zastosowania znacznika <fmt:formatNumber> do wyświetlenia wartości liczbowej jako wartości pieniężnej. W kroku (4) zmienna zawierająca datę zostaje wyświetlona w formacie pełnym, przy czym dokładna postać formatu zależy od ustawień językowych klienta HTTP.

Przykład wykorzystania znaczników 2/2

messages_en_US.properties

```
title=Good morning!
salary_label=Salary is:
date_label=Today is:
```

messages_fr.properties

```
title=Bonjour!
salary_label=Votre salaire:
date_label=Aujourd'hui nous sommes:
```

http://localhost:8084/Logika__biznesowa/fmttags.jsp

Plik Edycja Widok Ulubione Narzędzia Pomoc

Adres http://localhost:8084/Logika__biznesowa

Good morning!

Salary is: $1,200.00
Today is: July 26, 2006

http://localhost:8084/Logika__biznesowa/fmttags.jsp

Plik Edycja Widok Ulubione Narzędzia Pomoc

Adres http://localhost:8084/Logika__biznesowa/fmttags.jsp

Bonjour!

Votre salaire: 1 200,00 ¤
Aujourd'hui nous sommes: 26 juillet 2006

Pakiet plików z ustawieniami językowymi składa się z kilku plików o tej samej nazwie i rozszerzeniu *.properties, z dodanym sufiksem reprezentującym lokalizację językową. Przykładowo, pakiet messages wykorzystany w aplikacji składa się w rzeczywistości z dwóch plików: messages_en_US.properties (lokalizacja amerykańska) oraz messages_fr.properties (lokalizacja francuska). Oba pliki zawierają te same klucze, a komunikaty przypisane do każdego klucza są we właściwym języku. Serwer aplikacji pobiera lokalizację językową z klienta HTTP i zapewnia załadowanie właściwego pliku z ustawieniami językowymi. Zrzuty ekranu pokazują tę samą stronę JSP oglądaną z tej samej przeglądarki po zmodyfikowaniu preferowanego języka (w Microsoft IE: Narzędzia->Opcje internetowe->Języki).

Znaczniki JSTL XML

- <%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
- Znaczniki ogólnego przeznaczenia
  - **`<x:out> <x:set>`**
- Znaczniki do iteracji
  - **`<x:forEach>`**
- Znaczniki do wyboru warunkowego
  - **`<x:if> <x:choose> <x:when> <x:otherwise>`**
- Znaczniki do parsowania i transformacji
  - **`<x:parse> <x:transform>`**

Logika biznesowa (28)

Trzecia grupa znaczników JSTL to znaczniki do przetwarzania danych semistrukturalnych XML. W większości znaczniki te pokrywają się co do funkcjonalności ze znacznikami podstawowymi, nowość stanowią tylko znaczniki służące do parsowania i transformacji dokumentów XML.

<x:parse>: powoduje parsowanie dokumentu XML przekazanego przez atrybut doc i zapisanie przygotowanej wersji do zmiennej, odwołania do zawartości sparsowanego dokumentu XML odbywają się poprzez wywołania języka XPath

<x:transform>: umożliwia transformację źródłowego dokumentu XML za pomocą reguł transformacji XSLT zapisanych w osobnym szablonie

Aplikacje WWW

Przykład wykorzystania znaczników

```
<%@page contentType="text/html"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
  <body>
    <c:import var="xmlfile" url="students.xml"/>
    <x:parse doc="${xmlfile}" var="result"/>

    <x:forEach select="$result/students/student"
               var="currentStudent">
      <x:out select="$currentStudent/name"/><br/>
    </x:forEach>
  </body>
</html>
```

```
<students>
  <student id="1">
   <name>
     <first>John</first><last>Smith</last><middle>T</middle>
   </name>
   <grade> <points>88</points><letter>B</letter> </grade>
  </student>
...
```

Logika biznesowa (29)

Slajd przedstawia przykład wykorzystania znaczników do przetwarzania XML. Fragment pliku students.xml jest przedstawiony w dolnej części slajdu. W kroku (1) za pomocą znacznika <c:import> następuje wczytanie pliku XML do zmiennej xmlfile. W kroku (2) dokument XML zostaje przeanalizowany i sparsowany za pomocą znacznika <x:parse>, a wynikowe drzewo XML zostaje zapisane do zmiennej result. Krok (3) ilustruje wykorzystanie pętli <x:forEach> do iteracji po wszystkich elementach zwróconych przez zapytanie XPath. Podstawowa różnica między znacznikami podstawowymi <c:out>, <c:set>, itd. a ich odpowiednikami <x:out>, <x:set> polega na tym, że te ostatnie wspierają, poprzez atrybut select, język przeszukiwania XPath.

Znaczniki JSTL SQL

- <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

- Znacznik do definiowania połączenia
  - **`<sql:setDataSource>`**
- Znaczniki do obsługi zapytań SELECT
  - **`<sql:query> <sql:param> <sql:dateParam>`**
- Znaczniki do obsługi poleceń DML
  - **`<sql:update> <sql:param> <sql:dateParam>`**
- Znacznik do demarkacji transakcji
  - **`<sql:transaction>`**

Znaczniki JSTL służące do komunikacji z bazą danych są najbardziej krytykowanym i atakowanym zestawem znaczników JSTL. Wiąże się to z tym, że znaczniki te promują bardzo złe praktyki programistyczne, w szczególności powodują wymieszanie kodu odpowiedzialnego za łączenie się z bazą danych (znaczniki JSTL SQL) z kodem odpowiedzialnym za generowanie warstwy prezentacji. Faktycznie, należy unikać stosowania tych znaczników w dużych i złożonych projektach. Natomiast w przypadku niewielkich projektów, składających się z kilku powiązanych ze sobą stron JSP, wykorzystanie znaczników JSTL SQL bardzo przyspiesza i ułatwia tworzenie aplikacji internetowych wykorzystujących bazę danych. Innym powszechnym zastosowaniem znaczników JSTL SQL jest tworzenie rozwiązań testowych lub prototypowych. W ramach kategorii JSTL SQL mamy do dyspozycji następujące znaczniki.

<sql:setDataSource>: służy do wprowadzenia danych potrzebnych do nawiązania połączenia z bazą danych (nazwa użytkownika, hasło, sterownik JDBC, adres serwera, nazwa bazy danych, adres URL JDBC)

<sql:query>: wykonuje zapytanie i zapisuje wynik zapytania do zmiennej

<sql:param>: umożliwia parametryzację zapytania SQL poprzez późne wiązanie zmiennych

<sql:dateParam>: analogicznie do poprzedniego znacznika, w odniesieniu do parametrów będących typu DATE

<sql:update>: wykonuje polecenie DML (INSERT, UPDATE, DELETE)

<sql:transaction>: służy do demarkacji granic transakcji

Przykład zastosowania znaczników

```
<%@page contentType="text/html"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
  <body>
    <form> Lookup by jobs: <input type="text" name="empjob"/>          1
           <input type="submit" value="Lookup"/> </form>

    <sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
 2    user="scott" password="tiger" var="database" scope="page"
      url="jdbc:oracle:thin:@db.acme.com:1521:orcl"/>

    <sql:query var="employees" dataSource="${database}">
      SELECT * FROM emp WHERE job = ?
      <sql:param value="${param.empjob}"/>          3
    </sql:query>

      <c:forEach items="${employees.rows}" var="employee">
          <c:out value="${employee.ename}"/>,
     4    <c:out value="${employee.job}"/><br/>
      ...
```


Slajd przedstawia przykład wykorzystania znaczników JSTL SQL do stworzenia prostej strony umożliwiającej wprowadzanie kryteriów wyszukiwania i wyświetlanie wyników zapytań do bazy danych. Na początku strony mieści się formularz HTML (1) którego znacznik <form> nie posiada atrybutu action (zawartość formularza zostanie przekazana do bieżącej strony JSP). Krok (2) to zdefiniowanie zawartości znacznika <sql:setDataSource>. Oczywiście, wartości wpisywane do znacznika zależą wprost od wykorzystywanego sterownika JDBC, bazy danych, konta w bazie danych, itp. W kroku (3) zostaje wykonane zapytanie, do którego w momencie wykonania zostaje przekazany parametr (przez znacznik <sql:param>) odczytany z listy parametrów HTTP. Wreszcie w kroku (4) następuje iteracja po wszystkich obiektach zwróconych przez zapytanie i wyświetlenie nazwisk i zawodów pracowników na stronie.

Aplikacje WWW

Pliki znaczników

- Pisanie własnych znaczników przed JSP 2.0
  - procedura obsługi znacznika
  - definicja znacznika
  - opis biblioteki znaczników
- Od JSP 2.0 pliki znaczników
  - nie muszą być kompilowane
  - mogą być pisane w JSP
  - dostęp do obiektów **`request`**, **`response`**, itp.

Logika biznesowa (32)

Przed pojawieniem się specyfikacji JSP 2.0 tworzenie bibliotek własnych znaczników było możliwe, ale wiązało się z bardzo dużym nakładem pracy. Niezbędne było przygotowanie procedur obsługi każdego znacznika (otwarcie i zamknięcie znacznika, ciało znacznika, itp.), zdefiniowanie znacznika i powiązanie znaczników za pomocą plików *.tld z właściwymi procedurami obsługi każdego znacznika. Począwszy od wersji JSP 2.0 tworzenie własnych znaczników i bibliotek własnych znaczników uległo bardzo daleko idącemu uproszczeniu. Specyfikacja JSP 2.0 wprowadza pojęcie plików znaczników (ang. tag files). Plik znacznika przypomina wyglądem zwyczajną stronę JSP, może posiadać specjalne dyrektywy, skryptlety, ma dostęp do wielu obiektów predefiniowanych JSP, może też wykorzystywać język wyrażeń JSP EL. Tradycyjnie pliki znaczników mają rozszerzenie *.tag lub *.tagx, zaś współdzielone fragmenty plików znaczników mają rozszerzenie *.tagf. Pliki znaczników nie muszą (choć mogą) być kompilowane, nie wymagają plików deskryptorów *.tld i mogą być przygotowane tylko i wyłącznie za pomocą znaczników JSP. Oznacza to, że można tworzyć własne biblioteki znaczników bez konieczności jakiegokolwiek programowania. Pliki znaczników muszą być przechowywane w katalogu WEB-INF/tags/ lub dowolnym podkatalogu tego katalogu.

Pliki znaczników – przykład 1/2

biography.tag

```
<%@tag pageEncoding="UTF-8"%>   (1)

<%@attribute name="title" required="true"%>
<%@attribute name="person" required="true"%>          (2)
<%@attribute name="institution" required="true"%>
<%@attribute name="email" required="true"%>

<table border="0" width="300">
    <tr><td>${title} <b>${person}</b></td></tr>
    <tr><td>${institution}</td></tr>    (3)
    <tr><td><em>${email}</em></td></tr>
    <tr><td style="{font-family: Tahoma; font-size: small}">
      <jsp:doBody/>  td></tr>
</table>   (4)
```


Na slajdzie przedstawiono przykładowy plik znacznika <biography> umieszczony w pliku o nazwie biography.tag. W pierwszej linii umieszczona jest deklaracja wskazująca, że zawartość pliku definiuje nowy znacznik (1). W kroku (2) zdefiniowano atrybuty nowego znacznika, dla każdego atrybutu można podać typ atrybutu oraz określić, czy atrybut jest wymagany. Instrukcje w części (3) generują kod wynikowy znacznika, mają one dostęp zarówno do wartości atrybutów, ale także do wspomnianych wcześniej predefiniowanych obiektów JSP. Znacznik może także posiadać ciało. Instrukcja (4) powoduje włączenie ciała znacznika we wskazanym miejscu. Kolejny slajd przedstawia sposób wykorzystania pliku znacznika i wynik ostateczny.

Pliki znaczników – przykład 2/2

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib tagdir="/WEB-INF/tags" prefix="my"%>            1

<html>
  <body>
  <h1>Osoba</h1>
  <my:biography title="dr inż." person="Jan Kowalski"
      institution="Politechnika Poznańska"                   2
      email="Jan.Kowalski@put.poznan.pl" >
      Specjalizuje się w aplikacjach internetowych
      i pisaniu znaczników JSP. Najchętniej korzysta z
      mechanizmu plików                        nego          3
      wraz z wersją 2.
  </my:biography>
  </body>
</html>
```

Osoba

dr inż. **Jan Kowalski**

Politechnika Poznańska

Jan.Kowalski@put.poznan.pl

Specjalizuje się w aplikacjach internetowych i pisaniu znaczników JSP. Najchętniej korzysta z mechanizmu plików znaczników wprowadzonego wraz z wersją 2.0 specyfikacji JSP.

Na slajdzie przedstawiono sposób wykorzystania własnego znacznika zdefiniowanego uprzednio w pliku znacznika. Linia (1) jest obowiązkową dyrektywą wskazującą na katalog przechowujący pliki znaczników i nadającą wszystkim znacznikom wspólny prefiks. Bezpośrednio w kodzie strony JSP można umieścić zdefiniowany przez siebie znacznik (2), nie zapominając o atrybutach wymaganych. Część (3) stanowi ciało znacznika. Fragment zrzutu ekranu pokazuje końcowy efekt.

Łączenie plików znaczników w biblioteki

- Archiwum JAR z plikami znaczników
- Plik deskryptora biblioteki plików znaczników

```
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=http://java.sun.com/xml/ns/j2ee web-
jsptaglibrary_2_0.xsd  version="2.0">

  <description>JSP 2.0 tag files</description>
  <tlib-version>1.0</tlib-version>
  <short-name>My Tag Files</short-name>          1
  <uri>http://www.acme.com/tagfiles</uri>
  <tag-file>
    <name>biography</name>
    <path>/META-INF/tags/biography.tag</path>        2
  </tag-file>
</taglib>
```


Samodzielne pliki znaczników mogą być łączone w biblioteki. W tym celu należy utworzyć archiwum JAR zawierające wszystkie pliki znaczników. Pliki znaczników muszą znajdować się w katalogu META-INF/tags/. Dodatkowo, w katalogu META-INF wewnątrz archiwum JAR należy przygotować plik deskryptora biblioteki plików znaczników. Przykład pliku deskryptora przedstawiono na slajdzie. Poza standardowym nagłówkiem i wskazaniem na przestrzeń nazw plik deskryptora zawiera atrybuty opisujące bibliotekę (1): opis, numer wersji, skróconą nazwę, czy adres URI. Poza tym w pliku deskryptora każdemu plikowi znacznika z archiwum odpowiada jeden element <tag-file>, zawierający informację o nazwie i lokalizacji pliku znacznika (2). W celu użycia tak przygotowanej biblioteki plików znaczników wystarczy przekopiować archiwum JAR do katalogu WEB-INF/lib/ w systemie docelowym.

Materiały dodatkowe

- JavaBeans, http://java.sun.com/products/javabeans/
- JSTL, http://java.sun.com/products/jsp/jstl/
- Jakarta Taglibs, http://jakarta.apache.org/taglibs/index.html