

SOA – laboratorium nr 3

Temat: JSP

Wprowadzenie do technologii JSP

Strona JSP składa się ze statycznych (tekstowych) elementów i konstrukcji JSP, które tworzą część dynamiczną strony.

Strona JSP jest tłumaczona na serwlet. Wykonuje to za nas kontener WWW. Na początku więc – kod strony JSP tłumaczony jest na postać kodu źródłowego klasy Java (*.java), po czym następuje kompilacja na postać serwletu.

Dyrektywa `page` służy do importowania niezbędnych pakietów używanych na stronie JSP. Np.

```
package pakiet;  
public class licznik  
{  
    //ciało klasy  
    public static int getLiczba()  
    {  
        //ciało metody  
    }  
}
```

Aby teraz wykorzystać klasę Javy w naszym pliku JSP, wykonujemy

```
<%@ page import="pakiet.*"%>  
<html><body>  
<%  
    out.println( licznik.getLiczba() );  
%>  
</body></html>
```

Aby zaimportować wiele pakietów piszemy: `<%@ page import="pakiet.*, java.util.*"%>`.

Znaczek `<%@` - oznacza dyrektywę.

W powyższym przykładowym kodzie JSP – zbędne jest użycie metody `println`, wystarczy napisać: `<%= licznik.getLiczba() %>` - a wynik i tak będzie wyświetlony na stronie. Są to tzw. wyrażenia – tłumaczone one są na argumenty wywołań `out.print()`.

Skryptlet: `<% %>`

Dyrektywa `<%@ %>`

Wyrażenie `<%= %>`

Pomiędzy `<%!` i `%>` możemy deklarować w kodzie JSP zmienne i metody które są umieszczane w wygenerowanej klasie serwletu. Np.

```
<html><body>
<%!
    int podwojenie()
    {
        liczba = liczba * 2;
        return liczba;
    }
%>
<%!
    int liczba = 1;
%>
    Liczba wynosi:
<%=
    podwojenie()
%>
</body></html>
```

Biblioteka JSTL

Podstawowym celem standardowych bibliotek znaczników JSP (ang. *JSP Standard Tag Library* — JSTL) jest uproszczenie procesu tworzenia stron Java Server Pages. Skrypty prowadzą do powstawania nieutrzymywanych stron JSP i mogą być zastąpione przez standardowe akcje JSP. Standardowe akcje są jednak zbyt ograniczone i lepszym sposobem dla programistów Java jest tworzenie własnych niestandardowych akcji. Mimo to tworzenie akcji niestandardowych jest trudnym zadaniem. Biblioteka JSTL zapewnia takie akcje niestandardowe, które mogą obsługiwać typowe powtarzające się zadania. JSTL obejmuje szeroki zakres akcji podzielonych na poszczególne obszary funkcjonalne. W tabeli poniżej można znaleźć listę obszarów funkcjonalnych wraz z adresami URI odwołującymi się do bibliotek i przedrostkami stosowanymi w specyfikacji JSTL.

Obszar funkcjonalny	Adres URI	Przedrostek
Znaczniki podstawowe	http://java.sun.com/jsp/jstl/core	c
Przetwarzanie XML	http://java.sun.com/jsp/jstl/xml	x
Formatowanie obsługujące standard I18N (internacjonalizacja)	http://java.sun.com/jsp/jstl/fmt	fmt
Dostęp do relacyjnej bazy danych	http://java.sun.com/jsp/jstl/sql	sql
Funkcje	http://java.sun.com/jsp/jstl/functions	fn

Akcja	Opis
<c:catch>	Przechwytuje wyjątki generowane w sekcji body danej akcji.
<c:choose>	Wybiera jeden z wielu fragmentów kodu.
<c:forEach>	Przeprowadza iterację przez kolekcję obiektów lub iteruje stałą liczbę razy.
<c:forTokens>	Przeprowadza iterację przez tokeny w łańcuchu znaków.
<c:if>	Warunkowo wykonuje pewną funkcjonalność.
<c:import>	Importuje adres URL.
<c:otherwise>	Określa domyślną funkcjonalność w akcji <c:choose>.
<c:out>	Wysyła dane wyjściowe do bieżącego obiektu JspWriter.
<c:param>	Określa parametr adresu URL dla akcji <c:import> lub <c:url>.
<c:redirect>	Przekierowuje odpowiedź do określonego adresu URL.
<c:remove>	Usuwa zmienną objętą zakresem.
<c:set>	Tworzy zmienną objętą zakresem.
<c:url>	Tworzy adres URL z odpowiednim przepisywaniem adresu URL.
<c:when>	Określa jeden z kilku warunków akcji w <c:choose>.

JSTL – to stworzona przez kogoś i upakowana biblioteka znaczników, z których możemy w prosty sposób korzystać w naszych stronach JSP. Okazuje się że można również definiować swoje znaczniki.

Przykład w kodzie serwleta

```
String[] listaFilmow = { "Solaris", "Kroll", "Psy" };
request.setAttribute( "listaFilmow", listaFilmow );
```

...

Kod JSP:

```
<table>
<% String[] elementy = (String[])request.getAttribute( "listaFilmow" );
String zmienna = null;
for( int i=0 ; i<elementy.length ; i++ )
{
zmienna = elementy[ i ];
%>
<tr><td><%= zmienna %></td></tr>
<% } %>
</table>
```

Wykorzystując jednak bibliotekę znaczników nasz kod może wyglądać tak:

Kod JSP z wykorzystaniem JSTL:

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><body>
<strong>Lista filmów</strong>
<br>
<table>
7
<c:forEach var="film" items="{listaFilmow}">
<tr>
<td>{film}</td>
</tr>
</c:forEach>
</table>
</body></html>

```

Opis niektórych znaczników z biblioteki JSTL:

<c:forEach>

Jest on jakby odpowiednikiem pętli for. Powtarza on się dla każdego elementu czy to tablicy, odpowiedniej kolekcji tudzież łańcucha w postaci separatora. Znacznik ten możemy zagnieżdżać.

<c:if>

Odpowiednik kluczowego słowa if – dostępnego w większości języków programowania.

Znacznik <c:choose> oraz <c:when> i <c:otherwise>

Jest to jakby odpowiednik instrukcji swich, z tym że wybór może być tylko jeden.

```

<c:choose>
<c:when test="{preferencje == 'biały'}">
Lubisz kolor biały!
</c:when>
<c:when test="{preferencje == 'czerwony'}">
Lubisz kolor biały!
</c:when>
<c:otherwise>
Nie lubisz czerwonego i białego?
</c:otherwise>
</c:choose>

```

<c:set>

Znacznik ten służy np. do dodania nowego elementu dla klasy Map. (Nie możemy dodać elementów do list lub tablic).

<c:import>

Np. <c:import url="http://www.buldog.pl/index.html"> - czyli dodaje do bieżącej strony zawartość pliku wskazanego za pomocą atrybutu url. Przy czym mamy możliwość dołączania zasobów spoza obszaru kontenera.

Model MVC w oparciu o strony JSP

Zadanie 1

Celem ćwiczenia jest napisanie prostej aplikacji internetowej o nazwie *Doradca Piwny*. Aplikacja zbudowana zostanie zgodnie z prostą architekturą MVC w oparciu o strony JSP.

Aplikacja będzie posiadać dwie strony będące widokiem: *form.html* z formularzem wyboru koloru piwa, oraz stronę *wynik.jsp*, która wyświetla sugerowaną markę piwa na podstawie wyboru koloru. Kontroler aplikacji jest skonstruowany w postaci serwletu *WyborPiwa*. Modelem będzie komponent będącym klasą *EkspertPiwny*.

1. Utwórz nowy projekt dynamicznej strony internetowej w Eclipse pod nazwą *DoradcaPiwny*.
2. Utwórz stronę *form.html*, która będzie zawierać:
 - a. Formularz, którego metodą POST będzie uruchamiany serwlet o nazwie *WybierzPiwo.do*
 - b. Listę wyboru koloru piwa: jasny, bursztynowy, brązowy, ciemny (select, option)
 - c. Przycisk SUBMIT wysyłające żądanie do serwera
4. Przetestuj stronę form.html
5. Utwórz następujący serwlet:
 - a. Klasa com.example.web.WyborPiwa
 - b. Nazwa serwletu: Servlet_wybierzpiwo
 - c. Odwołanie URL (url-pattern)/WybierzPiwo.do

Utworzony teraz serwlet będzie kontrolerem. Zadaniem serwletu jest przyjmowanie parametrów żądania, wywoływanie odpowiedniej metody modelu, przekazanie stronie JSP informacji o wybranych piwach oraz wywołanie tej strony. Serwlet będzie tworzony w kilku etapach.

I wersja aplikacji (kontroler)

6. W funkcji *doPost* serwletu zamieść następujący kod:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("Porada piwna<br/>");
String c = request.getParameter("kolor");
out.println("<br/>Wybor kolor piwa: "+c);
```
7. Przetestuj działanie aplikacji.

Klasa modelu

Model traktowany jest jako "zaplecze" aplikacji. Model może być istniejącym już od dawna systemem informatycznym, dopiero teraz udostępnianym przez stronę www. Model nigdy nie jest ściśle związany z pojedynczą aplikacją internetową, więc kod powinien być umieszczany w jego własnych pakietach.

Założenia modelu:

- znajduje się w pakiecie com.example.model
 - udostępnia pojedynczą metodę *getMarki()*, która jako parametr pobiera kolor piwa i zwraca listę preferowanych marek piwa (ArrayList).
8. Napisz klasę *EkspertPiwny* będącym modelem aplikacji i w zależności od wybranego koloru piwa zwróć kilka przykładowych marek, np.

```
List marki = new ArrayList();
if (kolor.equals("bursztynowy") {
```

```
marki.add("Le**");
marki.add("Zy**ec");
}
```

II wersja aplikacji (model+kontroler)

9. Zmodyfikuj serwlet tak, aby utworzyć obiekt *EkspertPiwny*, wywołać metodę i zwrócić listę preferowanych piw.

10. Przetestuj aplikację.

Teraz w aplikacji brakuje jeszcze widoku *wynik.jsp*, gdzie ma być prezentowany ostateczny wynik.

III wersja aplikacji (model+widok+kontroler)

Kontroler należy zmodyfikować w taki sposób, aby przekazać do widoku odpowiedź uzyskaną od modelu a następnie wywołać widok *wyniki.jsp*.

11. W serwlecie, zamiast wypisywania wyniku w postaci kodu html, zamieść następujący kod:

```
request.setAttribute("marki", marki);
RequestDispatcher view = request.getRequestDispatcher("wyniki.jsp");
view.forward(request, response);
```

Kod ten dodaje do request-a atrybut o nazwie *marki* i przekazuje tam listę (zmienna *marki*) uzyskaną jako wynik działania metody w modelu. Następnie przekazuje żądanie do strony *wyniki.jsp*.

12. Utwórz stronę *wyniki.jsp* i wpisz tam następujący kod (zwróć uwagę na pogrubione linijki kodu):

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<c:forEach var="marka" items="${marki}">
${marka}<br/>
</c:forEach>
</body>
</html>
```

Do strony jsp dodana została biblioteka standardowa JSTL, która pozwala na tworzenie widoku w postaci dynamicznych stron jsp bez skryptów pisanych w Javie. Udostępnia ona takie polecenia jak if, for, switch itp. W tym przykładzie zastosowano pętlę, o strukturze ***<c:forEach>***, gdzie *items* przyjmuje

wartość tablicy, listy lub innej kolekcji. W tym przypadku jest to zmienna ukryta pod nazwą *marki*. Nazwa ta to nazwa atrybutu, który został przekazany do obiektu request w serwlecie. *Var* to nazwa zmiennej tymczasowej, która przechowuje kolejne iterowane obiekty pobierane z tablicy/listy/kolekcji. Znak \$ pozwala na bezpośredni dostęp do zmiennej o danej nazwie.

Nie zapomnij o ręcznym dodaniu biblioteki jstl-1.2.jar do projektu (sekcja lib). Plik możesz znaleźć w np. tutaj <https://mvnrepository.com/artifact/javax.servlet/jstl/1.2>.

14. Uruchom i przetestuj aplikację.

Rozszerz aplikację o weryfikację wieku użytkownika. Pojawia się dodatkowe pole wyboru wieku. W sytuacji gdy podany wiek wynosi poniżej 18 lat należy zamiast sugestii marki piwa które należy wybrać pojawić się tekst „Nieletnim nie wolno pić piwa”. Przy implementacji tego rozwiązania zastosuj filtry – znaną Ci z lab 2 koncepcję wspierającą serwlety. Zadaniem tego filtru jest nie dopuścić do przetwarzania zapytania przez kolejne klasy modelu MVC.

Zadania do samodzielnej realizacji

Zad 1. Zadanie polega na napisaniu czterech własnych znaczników JSP oraz serwletu zarządzającego sesją użytkownika.

- Servlet zarządzający sesją użytkownika powinien umożliwiać zalogowanie (na podstawie loginu i hasła) i wylogowanie użytkownika na podstawie żądań http. Login zalogowanego użytkownika powinien być zapisany w aktywnej sesji.
- Formularz zalogowania/wylogowania użytkownika zrealizowany jako tag JSP zdefiniowany w pliku .tag. W zależności od stanu sesji tag powinien wyświetlać formularz logowania (login i hasło) lub formularz wylogowania. Formularz powinien wywoływać odpowiednie akcje w servlecie zarządzającym sesją.
- Tag wyświetlający listę wszystkich aktywnych użytkowników z datą ich zalogowania posortowaną alfabetycznie. Lista użytkowników powinna być przechowywana w beanie o odpowiednim zasięgu a sam tag powinien być zrealizowany jako plik .tag oraz przyjmować dwa atrybuty definiujące czy lista ma być posortowana rosnąco/malejąco (wymagany) oraz kolor wyświetlenia daty logowania (opcjonalny)
- Tag formatujący wyświetlanie akapitu tekstu zrealizowany jako klasa tag handler. Tag powinien korzystać z trzech atrybutów definiujących treść nagłówka akapitu (wymagany), wyrównanie (prawa/lewa strona) nagłówka (opcjonalny) oraz kolor treści akapitu (opcjonalny). Tekst akapitu znajduje się w zawartości znacznika.
- Funkcja statyczna realizująca przeliczanie walut i zaokrąglająca wynik do dwóch miejsc po przecinku. Funkcja powinna przyjmować trzy argumenty, wartość wejściową, walutę wejściową i walutę wyjściową. Kursy walut (przynajmniej czterech) powinny być przechowywane w plikach properties.

Zadanie 2 - Księga gości

Przygotuj **program** obsługujący prostą księgę gości. Założenia:

- Dane podane przez użytkownika powinny być utrwalane w pamięci serwera - zalecana struktura to *Vector*.
 - Dane nie muszą być utrwalane na stałe. Po restarcie serwera mogą być zerowane.

- Dane wpisane w formularzu powinny być widoczne również z innej sesji przeglądarki (proszę przeprowadzić testy w przeglądarce trybie "Prywatnym").\

Przykładowy wygląd formularza:

Please submit your feedback:
Your name:
Your email:
Comment:

Formularz po dwukrotnym wypełnieniu:

Please submit your feedback:
Your name:
Your email:
Comment:

Jan Kowalski (jan@kowalski.pl) says
Tak, to ja!

Janina Kowalska (janina@kowalska.pl) says
Jana nie znam

- a. Utworzenie strony JSP, która będzie wyświetlała zawartość zbioru z wiadomościami (zbiór może być wpisany na stałe do kodu źródłowego strony JSP).
 - b. Utworzenie strony JSP, która będzie odczytywała i wyświetlała wiadomości zapisane do pliku przy pomocy programu utworzonego na poprzednich zajęciach.
3. Utworzenie stron JSP, która pozwoli na dodawanie wiadomości do pliku z wykorzystaniem formularza WWW.
 4. Utworzenie stron JSP, które pozwolą na edycję zapisanych wiadomości przy czym daną pozycję może edytować wyłącznie jedna osoba na raz.


Zadanie 3.

Zadanie polega na zaimplementowaniu gry w zgadywanie liczby. Program składa się z jednej strony strony JSP stanowiącej interfejs użytkownika i źródło logiki programu. Strona JSP zawiera formularz HTML umożliwiający podanie dowolnej liczby. Gra przebiega w następujący sposób: przed rozpoczęciem gry następuje wylosowanie liczby-zagadki z przedziału $<0, 100>$. Następnie gracz podaje liczbę z przedziału $<0,100>$ używając formularza HTML. W odpowiedzi gracz jest informowany czy podana liczba jest większa lub mniejsza od liczby zagadki. Gracz kontynuuje grę do momentu odgadnięcia zagadkowej liczby. Po odgadnięciu liczby-zagadki gracz jest informowany o liczbie wykonywanych prób. Liczba-zagadka oraz liczba prób gracza są przechowywane w komponentach warstwy backowej aplikacji i przekazywane w zasięgu **request**.



Podaj liczbę:

Twoja liczba (50) jest **większa** niż zagadka



Podaj liczbę:

Brawo, zgadła(e)ś po 6 próbach.
Spróbuj [raz jeszcze](#).

Zadanie 4.

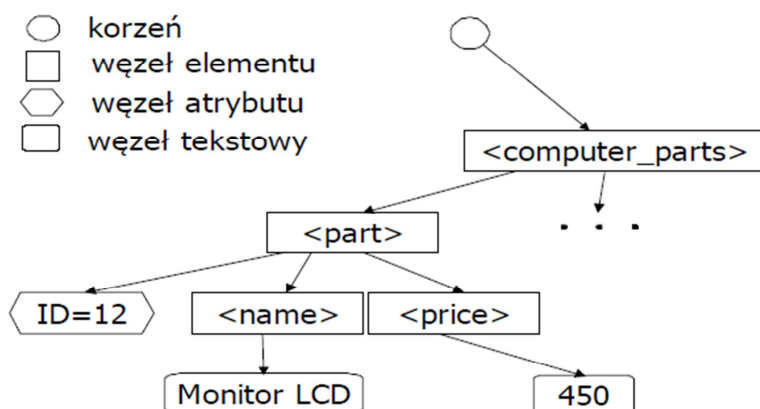
Proszę napisać prosty sklep internetowy przy użyciu stron JSP i komponentu Java. Biblioteka JSTL zostanie wykorzystana do przetwarzania pliku XML, sterowania przepływem pracy oraz wyświetlania danych.

Budowany sklep internetowy będzie oferował funkcję koszyka, w którym klient może umieszczać (lub z niego usuwać) produkty. Ze strony osoby programującej wymaga to zapewnienia, że informacje o produktach wybranych przez klienta nie będą ulotne. W tym celu można zastosować komponent JavaBean (o zasięgu sesji) reprezentujący koszyk zakupów.

Klasa będzie oferowała możliwość przechowywania identyfikatorów produktów w koszyku wraz z liczbą sztuk danego produktu. Takie dane można w łatwy sposób składować w tablicy haszowej, której kluczem jest identyfikator produktu, a wartością liczba zamówionych sztuk tego produktu. Ponadto, dostępne są metody do dodawania produktu (można dodawać po jednej sztuce), usuwania produktu (niezależnie od liczby sztuk) oraz pobierania liczby różnych produktów i całej zawartości koszyka.

Dane na temat towarów oferowanych w sklepie będą znajdowały się w pliku XML. Plik zawiera opis części komputerowych. Struktura drzewa DOM (z przykładowymi

wartościami) tego dokumentu została przedstawiona poniżej.



Na głównej stronie sklepu internetowego będą wyświetlone nazwy i ceny produktów, które zapisane są w pliku XML. Oprócz tego powinna istnieć strona służąca do zarządzania zawartością koszyka.

Zadanie 5.

Przygotuj stronę JSP o nazwie strona.jsp, która odczyta zawartość listy filmów i wyświetli ją w postaci tabelarycznej. Posłuż się znacznikami `<c:forEach>`, `<c:out>`, `<c:choose>`, `<c:when>` i `<c:otherwise>` do przygotowania strony. Zwróć uwagę, że dochody uzyskiwane przez filmy są wyświetlone jak pieniądze, posłuż się znacznikiem `<fmt:formatNumber>` do poprawnego sformatowania tych wartości. Jeśli gatunek filmu to „wojenny”, zmień tło komórki tabeli na inny kolor. Twoja aplikacja powinna wyglądać następująco:

Lista filmów			
Tytuł	Gatunek	Rok	Dochód
Ojciec chrzestny	dramat	1972	\$120,000,000.00
Pluton	wojenny	1986	\$50,000,000.00
Nagi instynkt	thriller	1992	\$100,000,000.00