JSP i Servlety

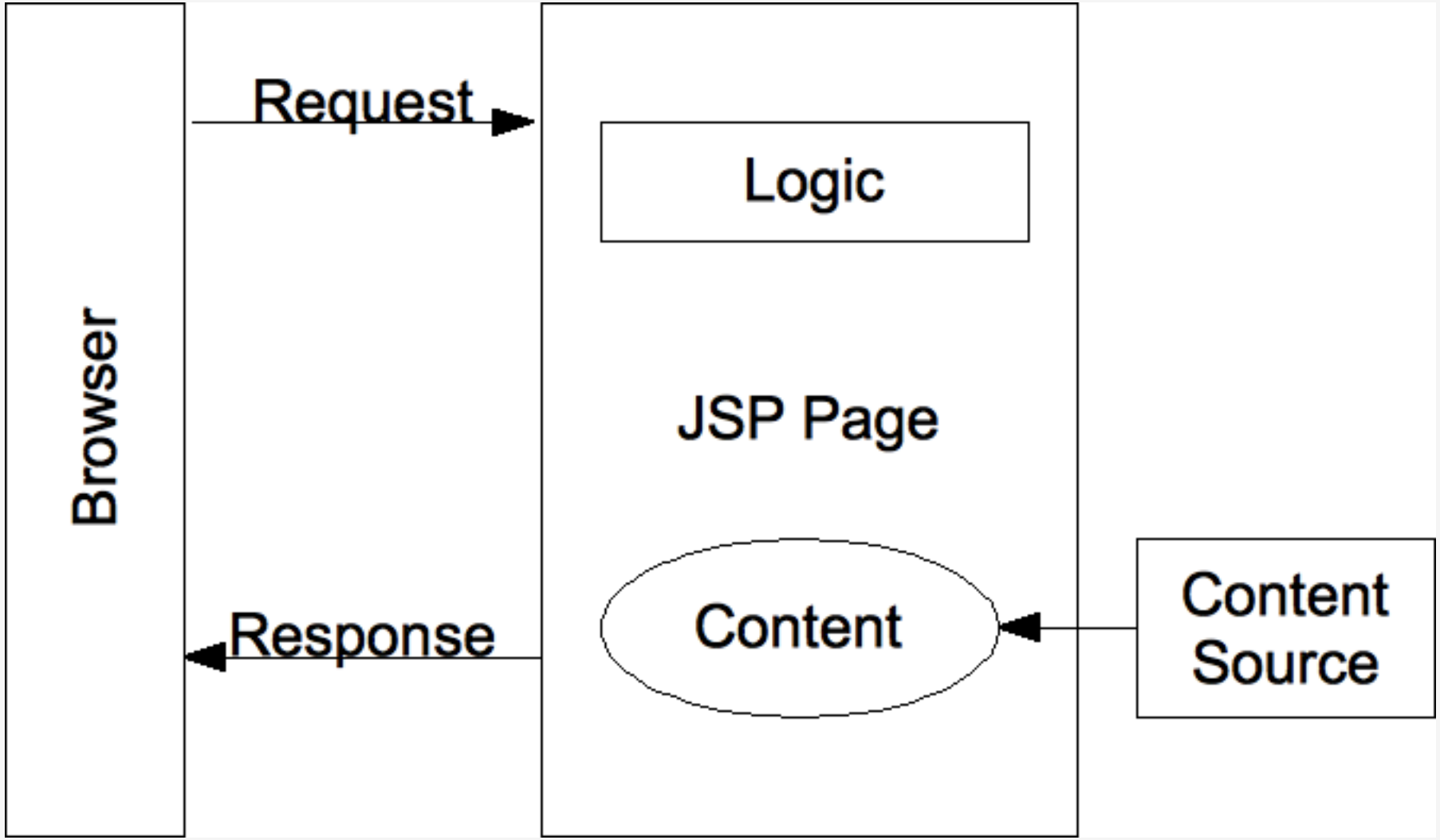
Rafał Roppel

JSP

JSP - Wprowadzenie do technologii JSP

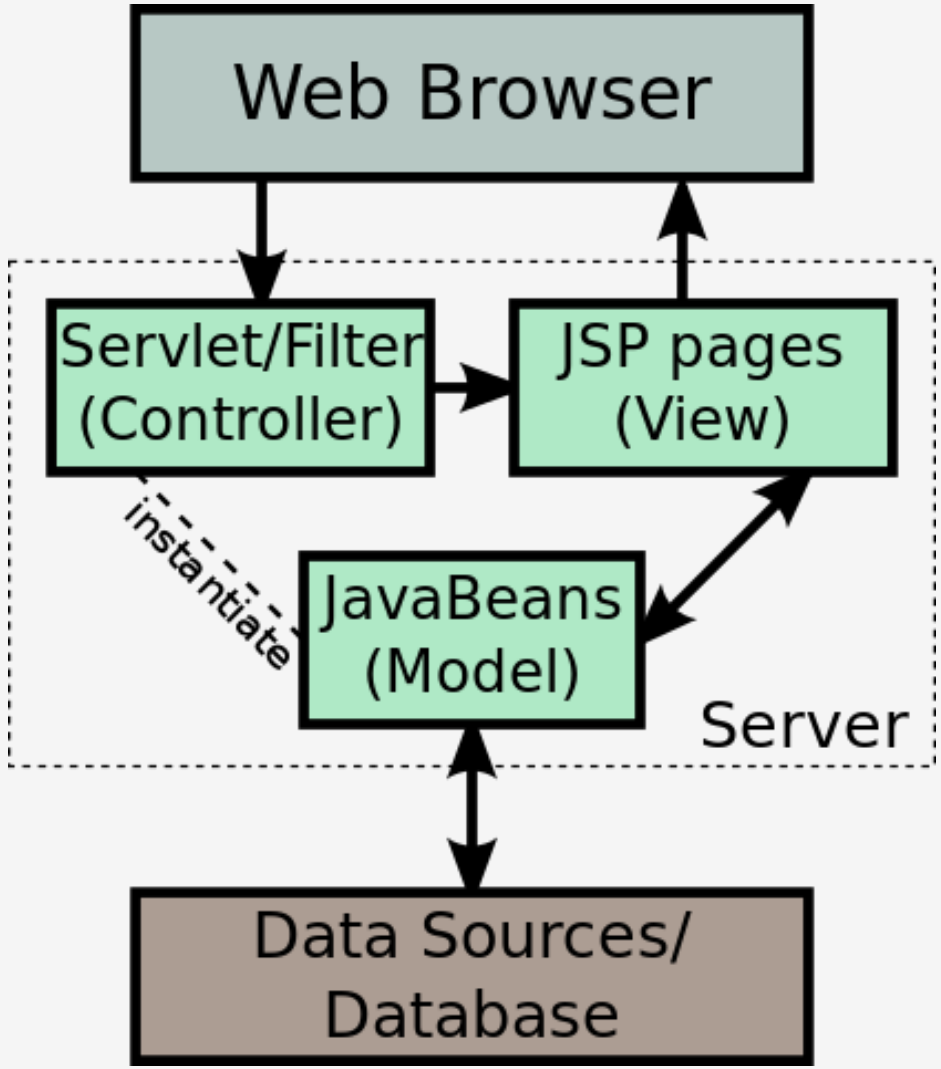
- Technologia JavaServer Pages jest integralną częścią J2EE - platformy aplikacji klasy enterprise wprowadzoną w 1999 roku
- Umożliwia generowanie dynamicznych stron internetowych
- Kod stron JSP jest w większości kodem HTML
- Dodatkowo posiada znaczniki JSP i JSTL oraz kod Javowy, które „ożywiają” stronę
- Dzięki temu (m.in.) separuje się warstwę prezentacji od warstwy logiki biznesowej, a kod HTML może być przygotowywany oddzielnie przez webmastera przy użyciu dedykowanych narzędzi
- Kod strony JSP konwertowany jest (automatycznie) do servletu
- Do uruchomienia strony JSP wymagają serwera webowego, np. Tomcata
- Zalecane rozszerzenia plików JSP, to: .jsp, .jspx, .jspf

JSP - Model 1



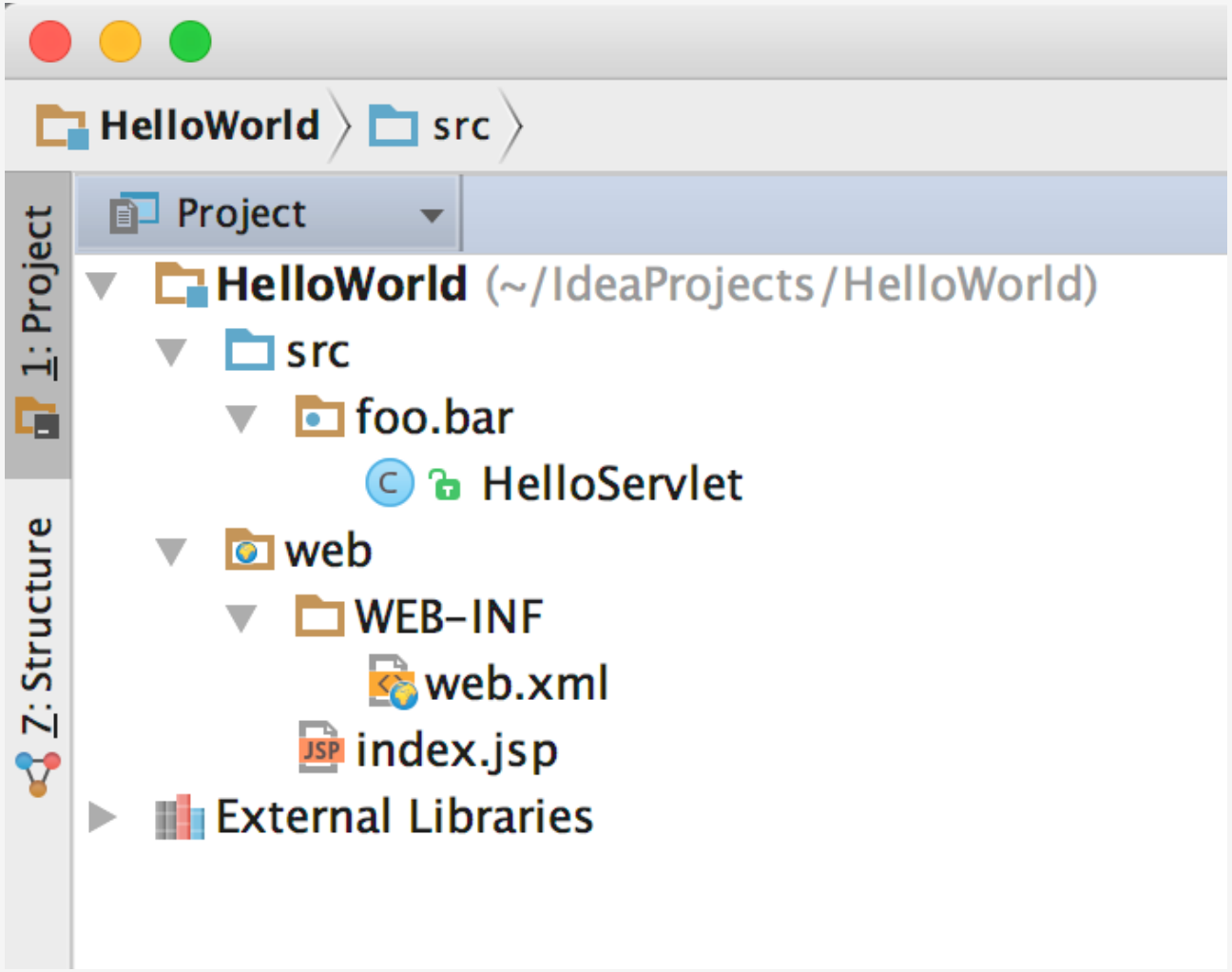
https://upload.wikimedia.org/wikipedia/commons/f/ff/Model_1.png

JSP - Model 2

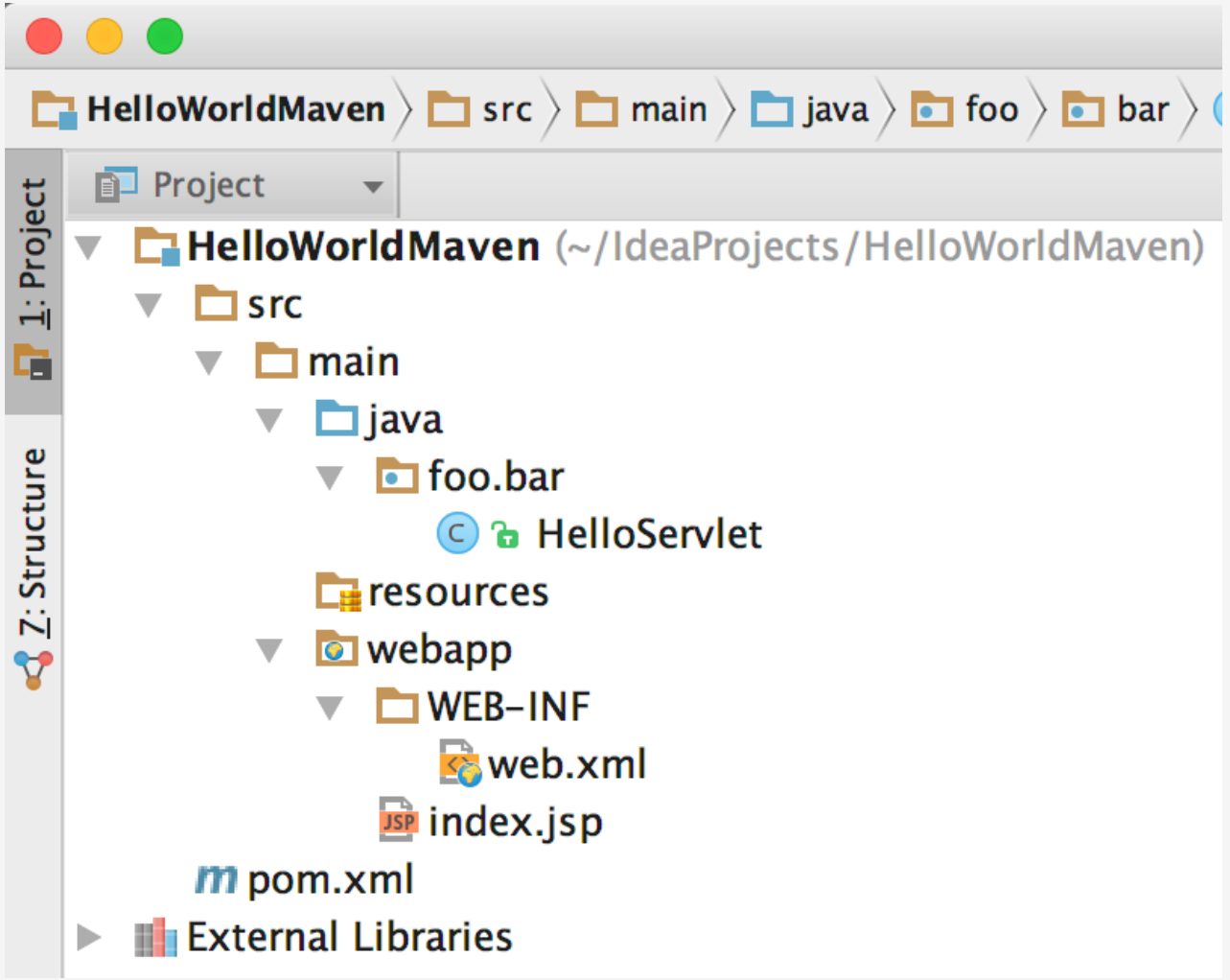


https://upload.wikimedia.org/wikipedia/commons/thumb/7/72/JSP_Model_2.svg/419px-JSP_Model_2.svg.png

JSP - Struktura projektu WEB



JSP - Struktura projektu WEB (maven)



JSP - Hello World

```jsp
<%@page language="java" contentType="text/html" %>
<html>
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1><% out.print("Hello World!");%></h1>
    </body>
</html>
```

Ćwiczenie

Hello World!

JSP - Cykl życia strony JSP

- Kompilacja
  - parsowanie
  - konwersja do servletu
  - kompilacja servletu
- Inicjalizacja
  - wywołanie metody jsp**Init**()
- Przetwarzanie żądań
  - wywoływanie metody _jsp**Service**(request, response)
- Zwalnianie zasobów
  - wywołanie metody jsp**Destroy**()

JSP - Przetwarzanie żądania

1. Przeglądarka wysyła żądanie (*request*) HTTP do serwera
2. Serwer rozpoznaje żądanie strony JSP i przekazuje sterowanie do silnika JSP
3. Silnik JSP zaczytuje stronę JSP z dysku i konwertuje ją do servletu
4. Kod servletu jest kompilowany i uruchamiany
5. Servlet przygotowuje odpowiedź w formacie HTML
6. Kod HTML przekazywany jest do serwera
7. Serwer odsyła odpowiedź (*response*) HTTP do przeglądarki

Konwersja i kompilacja z punktów 3-4 najczęściej zachodzi podczas instalacji aplikacji na serwerze, tak aby pierwsze żądanie nie było obarczone koniecznością oczekiwania na te operacje

JSP - Zmienne zdefiniowane (1)

- **request** (ServletRequest) - używany do pobierania informacji o żądaniu (parametrów, atrybutów)
- **response** (ServletResponse) - reprezentuje odpowiedź HTTP; umożliwia wysłanie nagłówka, ustawienie ciastka
- **session** (HttpSession) - trzyma informacje o sesji użytkownika
- **out** (JspWriter) - służy do wypisywania tekstu na stronie

JSP - Zmienne zdefiniowane (2)

- **application** (ServletContext) - przechowuje atrybuty związane z servletem dostępne dla wszystkich użytkowników
- **config** (ServletConfig) - przechowuje parametry inicjalizujące
- **pageContext** (PageContext) - to samo co jspContext oraz dodatkowe metody dedykowane servletom
- **page** (Object) - zwraca referencje do aktualnej strony JSP, synonim this

JSP - Elementy strony - wyrażenia

- Wyrażenia **<%= expression %>**
  - ich wartość zostanie obliczona i wstawiona do danych wyjściowych servletu
  - kontener pobiera zawartość pomiędzy <%= a %> i umieszcza jako argument metody print() klasy JspWriter (PrintWriter): out.print()
  - UWAGA: wyrażenie nie może kończyć się średnikiem!

```
<html>
  <head>
    <title>Dzisiaj jest...</title>
  </head>
  <body>
    <p>Dzisiaj jest: <%= java.time.LocalDate.now().toString()%></p>
  </body>
</html>
```

Ćwiczenia

Zmienne zdefiniowane

JSP - Elementy strony - skryptlety

- Skryptlety **<% code %>**
  - mogą zawierać dowolną ilość kodu Javy
  - oferują dostęp do zmiennych zdefiniowanych dla stron JSP
  - mogą służyć np, do ustawiania nagłówków odpowiedzi oraz kodów stanu, komunikacji z bazą danych i innych złożonych instrukcji

```
<html>
    <head>
        <title>Dzisiaj jest...</title>
    </head>
    <body>
        <p>Dzisiaj jest: <% out.println(java.time.LocalDate.now().toString());%></p>
    </body>
</html>
```

JSP - Elementy strony - dyrektywy (1)

- Dyrektywy **<%@ directive attribute=„value”... %>**
  - kontrolują proces translacji strony
  - **page** - atrybuty związane ze stroną
    - **import** - pozwala na określenie klas i pakietów, które zostaną zaimportowane do servletu wygenerowanego ze strony JSP
    - **info** - definiuje, co zwróci metoda `getServletInfo()`
    - **isELIgnored** - określa, czy ignorować wyrażenia EL
    - **isScriptingEnabled** - określa, czy ignorować elementy skryptowe

JSP - Elementy strony - dyrektywy (2)

- **pageEncoding** - specyfikuje kodowanie strony
- **contentType** - określa nagłówek odpowiedzi o nazwie Content-Type
- **isThreadSafe** - wskazuje, czy wygenerowany servlet ma implementować interfejs SingleThreadModel
- **session** - definiuje, czy podłączać stronę pod sesję HTTP
- **extends** - określa, po której klasie servlet ma dziedziczyć

JSP - Elementy strony - dyrektywy (3)

- **buffer** - definiuje rozmiar bufora wykorzystywanego przez zmienną out klasy `JspWriter`
- **autoflush** - kontroluje, czy bufor przechowujący dane wyjściowe powinien być automatycznie wysyłany w przypadku przepełnienia, czy generowany ma być wyjątek
- **errorPage** - wskazuje stronę JSP, która powinna obsłużyć wyjątek rzucony (i nie obsłużony) na bieżącej stronie
- **isErrorPage** - określa, że bieżąca strona jest stroną obsługi błędów występujących na innych stronach

JSP - Elementy strony - dyrektywy (4)

- **include**
  - <%@include file=„embedded.jspf”%>
  - pozwala dołączyć plik, np. fragment (segment) strony JSP
  - dołączana strona nie musi być *well-formed*, ponieważ nie będzie przetwarzana przed załączeniem
- **taglib**
  - <%@taglib uri=„http://foo.com/ftags” prefix=„f”%>
  - deklaruje użycie biblioteki tagów

JSP - Elementy strony - deklaracje

- Deklaracje **<%! code %>**
  - pozwalają na zdefiniowanie metod oraz pól, które zostaną wstawione do głównego ciała klasy servletu
  - używane zazwyczaj z wyrażeniami lub skryptletami

```
<%@page language="java" contentType="text/html; charset=UTF-8" %>
<%@page pageEncoding="UTF-8" %>

<%! private long visitCount = 0; %>

<h2>Ilość odwiedzin strony: <%= ++visitCount%></h2>
```

JSP - Elementy strony - komentarze (1)

- Komentarze **<%— text —%>**
  - służą do umieszczania dodatkowych informacji na temat kodu pisanego w JSP
  - tekst komentarza nie będzie dostępny w kodzie źródłowym servletu wygenerowanego ze strony JSP ani w kodzie HTML w przeglądarce - w odróżnieniu od komentarza HTML

```
<%@page language="java" contentType="text/html; charset=UTF-8" %>
<%@page pageEncoding="UTF-8" %>

<html>
<head>
    <title>Komentarze JSP i HTML</title>
</head>
<body>
<h1>Komentarze JSP i HTML</h1>
<%-- Komentarz JSP --%>
<!-- komentarz HTML -->
</body>
</html>
```

JSP - Elementy strony - komentarze (2)

```
out.write("    <html>\n");
out.write("    <head>\n");
out.write("        <title>Komentarze JSP i HTML</title>\n");
out.write("    </head>\n");
out.write("    <body>\n");
out.write("    <h1>Komentarze JSP i HTML</h1>\n");
out.write("    ");
out.write("\n");
out.write("    <!-- komentarz HTML -->\n");
out.write("    </body>\n");
out.write("    </html>\n");
```

```
<html>
<head>
    <title>Komentarze JSP i HTML</title>
</head>
<body>
<h1>Komentarze JSP i HTML</h1>

<!-- komentarz HTML -->
</body>
</html>
```

Ćwiczenia

Skryptlety, dyrektywy, deklaracje, komentarze

JSP - Elementy strony - język wyrażeń (1)

- Język wyrażeń
  - `${wyrażenie}`
    - natychmiastowa ewaluacja
    - tylko do odczytu
    - obliczane po pierwszym wyświetleniu strony
  - `#{wyrażenie}`
    - do odczytu i zapisu
    - obliczane w dowolnym etapie cyklu życia strony
- Można ich używać w statycznym tekście oraz w atrybutach znaczników, które to umożliwiają
- Wyrażenie jest automatyczne konwertowane do potrzebnego typu

JSP - Elementy strony - język wyrażeń (2)

- Dostęp do predefiniowanych zmiennych

applicationScope	Map<String, Object>
cookie	Map<String, Cookie>
header	Map<String, String>
headerValues	Map<String, String[]>
initParam	Map<String, String>
pageContext	PageContext
pageScope	Map<String, Object>
param	Map<String, String>
paramValues	Map<String, String[]>
requestScope	Map<String, Object>
sessionScope	Map<String, Object>

JSP - Elementy strony - znaczniki akcji (1)

- Znaczniki akcji **<jsp:action attribute=„value”...></jsp:action>**
- Mogą być zagnieżdżone
- Są przetwarzane podczas żądania strony i umożliwiają wykonanie poniższych operacji:
- **forward**
  - <jsp forward page=„other.jsp”/>
  - przerywa przetwarzanie bieżącej strony i przekazuje request do wskazanej strony
- **include**
  - <jsp:include page=„footer.jsp”/>
  - dołącza stronę JSP do bieżącej strony; dołączana strona musi być *well-formed,* ponieważ dołączany jest wygenerowany przez nią kod HTML
- **param**
  - <jsp:param name=„id” value=„5”/>
  - pozwalają dodać parametry dla strony (servletu) wywoływanych przy użyciu jsp:forward lub jsp:include

JSP - Elementy strony - znaczniki akcji (2)

- **useBean**
  - <jsp:useBean id=„name” class=„package.class” type=„object type” scope=„page|request|session|application”/>
  - umożliwia korzystanie ze standardowych JavaBeans
- **setProperty**
  - <jsp:setProperty name=„beanId” property=„beanPropertyName” value=„value to set”/>
  - umożliwia ustawienie wartości pola beana; bean musi być wcześniej zdefiniowany poprzez <jsp:useBean.../>
- **getProperty**
  - <jsp:getProperty name=„beanId” property=„beanPropertyName”/>
  - umożliwia pobranie wartości pola beana

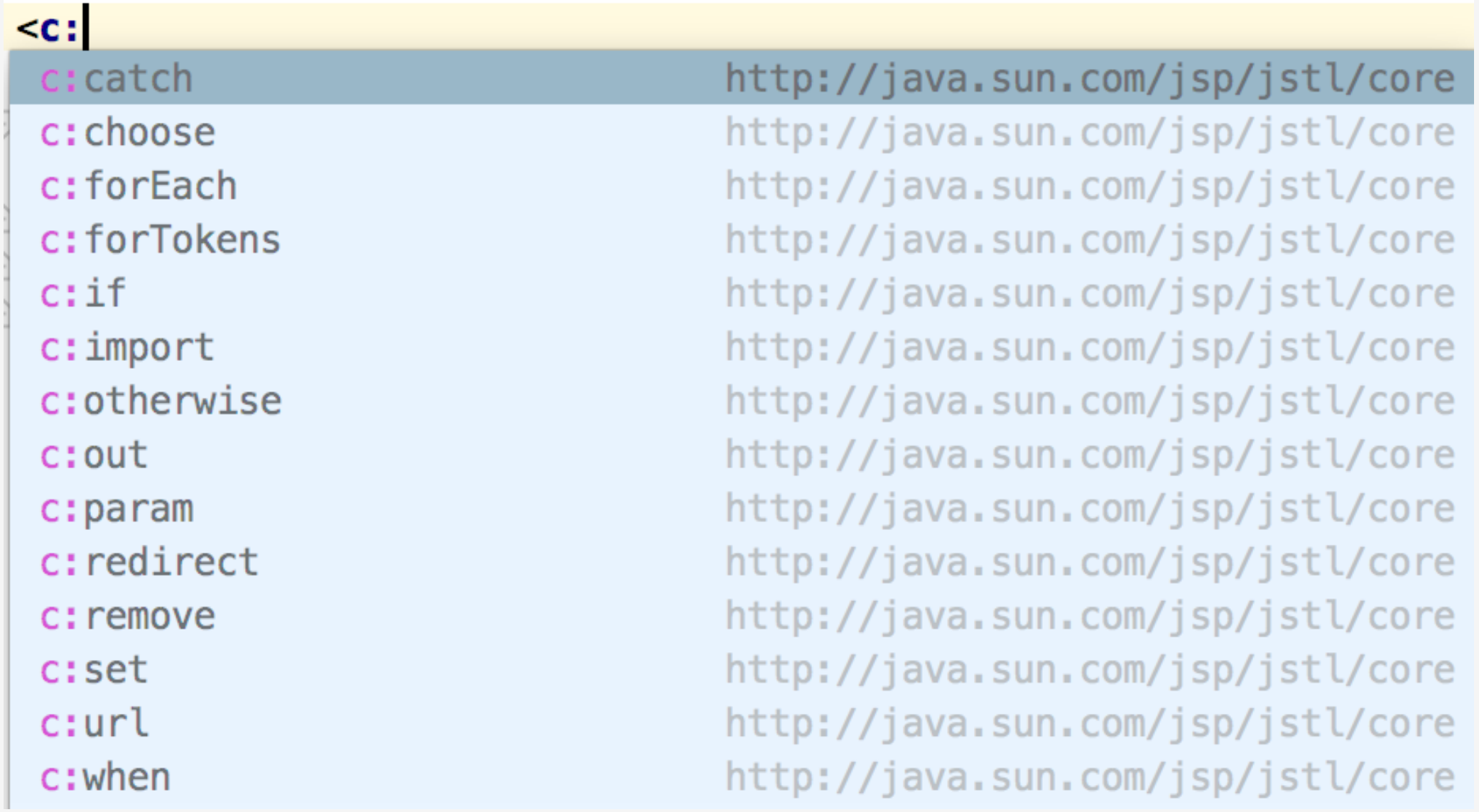
Ćwiczenia

Znaczniki akcji

JSP - JSTL

- JavaServer Pages Standard Tag Library (JSTL), to zbiór użytecznych tagów JSP, które udostępniają wiele funkcjonalności wykorzystywanych przy pisaniu stron JSP
- Aby móc korzystać z biblioteki tagów, należy zainkludować ją przy użyciu dyrektywy taglib
- Popularne biblioteki
  - Core tags
  - Formatting tags
  - SQL tags
  - XML tags
  - JSTL functions

JSP - JSTL - Core tags



JSP - JSTL - Formatting tags

```
<fmt:
```

fmt:bundle	http://java.sun.com/jsp/jstl/fmt
fmt:formatDate	http://java.sun.com/jsp/jstl/fmt
fmt:formatNumber	http://java.sun.com/jsp/jstl/fmt
fmt:message	http://java.sun.com/jsp/jstl/fmt
fmt:param	http://java.sun.com/jsp/jstl/fmt
fmt:parseDate	http://java.sun.com/jsp/jstl/fmt
fmt:parseNumber	http://java.sun.com/jsp/jstl/fmt
fmt:requestEncoding	http://java.sun.com/jsp/jstl/fmt
fmt:setBundle	http://java.sun.com/jsp/jstl/fmt
fmt:setLocale	http://java.sun.com/jsp/jstl/fmt
fmt:setTimeZone	http://java.sun.com/jsp/jstl/fmt
fmt:timeZone	http://java.sun.com/jsp/jstl/fmt

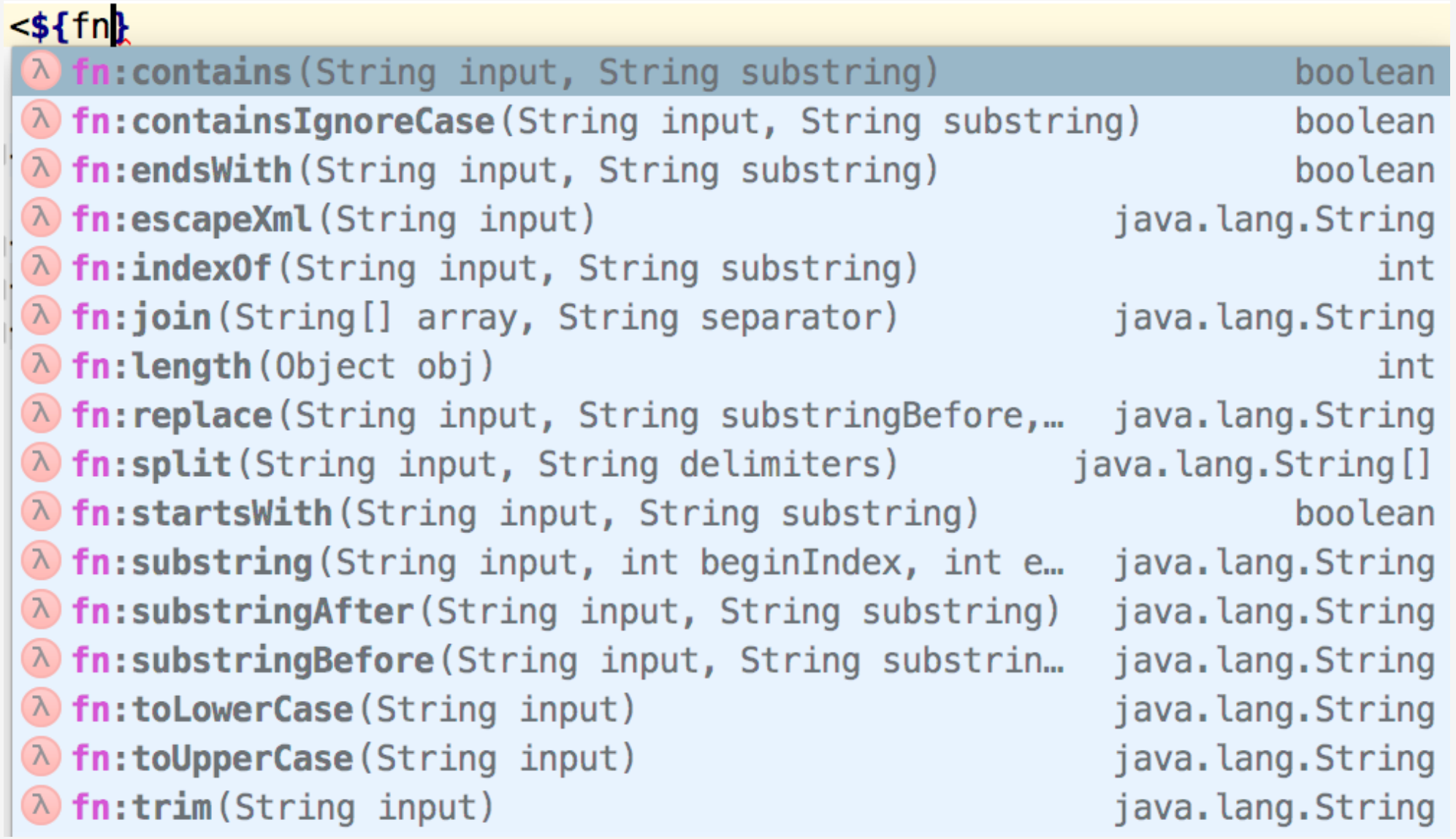
JSP - JSTL - SQL tags

```
<sql:
```

```
sql:dateParam          http://java.sun.com/jsp/jstl/sql
sql:param              http://java.sun.com/jsp/jstl/sql
sql:query              http://java.sun.com/jsp/jstl/sql
sql:setDataSource      http://java.sun.com/jsp/jstl/sql
sql:transaction        http://java.sun.com/jsp/jstl/sql
sql:update             http://java.sun.com/jsp/jstl/sql
```

JSP - JSTL - XML tags

JSP - JSTL - Functions



JSP - Client Request

- Obiekt requestu zawiera wszystkie informacje związane z żądaniem użytkownika
- W protokole HTTP dane te przesyłane są poprzez nagłówki (headers) i treść wiadomości (message body)
- HttpServletRequest
  - treść wiadomości
  - parametry
  - atrybuty
  - nagłówki
  - ścieżki
  - ciasteczka
  - kodowanie

```
iMac:~ rafos$ telnet localhost 8080
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.1
Host: localhost:8080
User-Agent: curl/7.43.0
Accept: */*
```

JSP - Client Response

- Obiekt response zawiera wszystkie informacje związane z odpowiedzią servera do klienta
- W protokole HTTP dane te przesyłane są poprzez nagłówki (headers) i treść wiadomości (message body)
- HttpServletResponse
  - treść odpowiedzi
  - nagłówki
  - ciasteczka
  - kodowanie

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=D8825E65DC9681319B83935EBB05BAB1; Path=/; HttpOnly
Content-Type: text/html;charset=UTF-8
Content-Length: 54
Date: Fri, 12 Feb 2016 23:38:23 GMT



<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

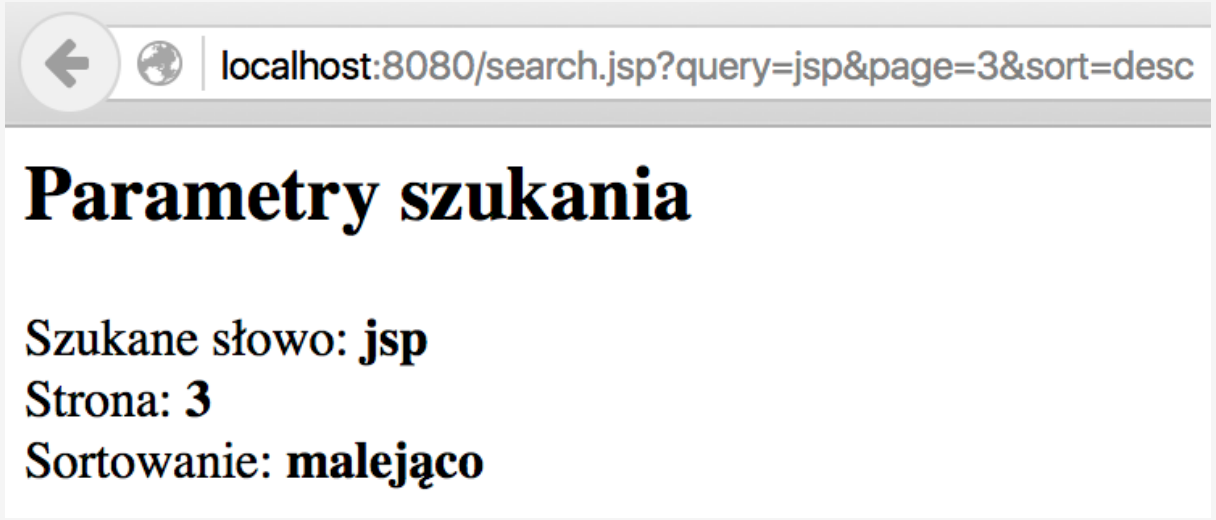
JSP - HTTP Status Codes

kod	tekst	opis
200	OK	Zawartość żądanego dokumentu
201	Created	Utworzono - wysłany dokument został zapisany na serwerze
301	Moved Permanently	Trwale przeniesiony - żądany zasób zmienił swój URI i w przyszłości zasób powinien być szukany pod nowym wskazanym adresem
302	Found	Znaleziono - żądany zasób jest chwilowo dostępny pod innym adresem
307	Temporary Redirect	Tymczasowe przekierowanie - żądany zasób znajduje się chwilowo pod innym adresem URI, odpowiedź powinna zawierać ten adres
400	Bad Request	Nieprawidłowe zapytanie - żądanie nie może być obsłużone przez serwer z powodu błędnej składni zapytania
401	Unauthorized	Nieautoryzowany dostęp - żądanie zasobu, który wymaga uwierzytelnienia
403	Forbidden	Zabroniony - serwer zrozumiał zapytanie lecz konfiguracja bezpieczeństwa zabrania mu zwrócić żądany zasób
404	Not Found	Nie znaleziono - serwer nie odnalazł zasobu według podanego URL
500	Internal Server Error	Wewnętrzny błąd serwera - serwer napotkał niespodziewane trudności
503	Service Unavailable	Usługa niedostępna - serwer nie jest w stanie w danej chwili zrealizować zapytania klienta ze względu na przeciążenie

JSP - Metoda GET (1)

- http://localhost:8080/search.jsp?query=jsp&page=3&sort=desc
- Domyślna metoda wysyłania informacji z przeglądarki do serwera
- Przesyłane parametry dołączane do żądania po znaku ?
- Poszczególne serwery wprowadzają (konfigurowalny) limit na długość żądania
- Parametry GET żądania dostępne są po stronie serwera w zmiennej QUERY_STRING
- Poszczególne parametry dostępne są za pomocą metod
  - request.getParameter(„nazwaParametru”)
  - request.getParameterNames()
  - request.getParameterMap()

JSP - Metoda GET (2)




```
<body>
<h2>Parametry szukania</h2>
<p>
    Szukane słowo: <b><%=request.getParameter("query")%></b><br/>
    Strona: <b><%=request.getParameter("page")%></b><br/>
    Sortowanie: <b><%=("desc".equals(request.getParameter("sort")) ? "malejąco" : "rosnąco")%></b><br/>
</p>
</body>
```

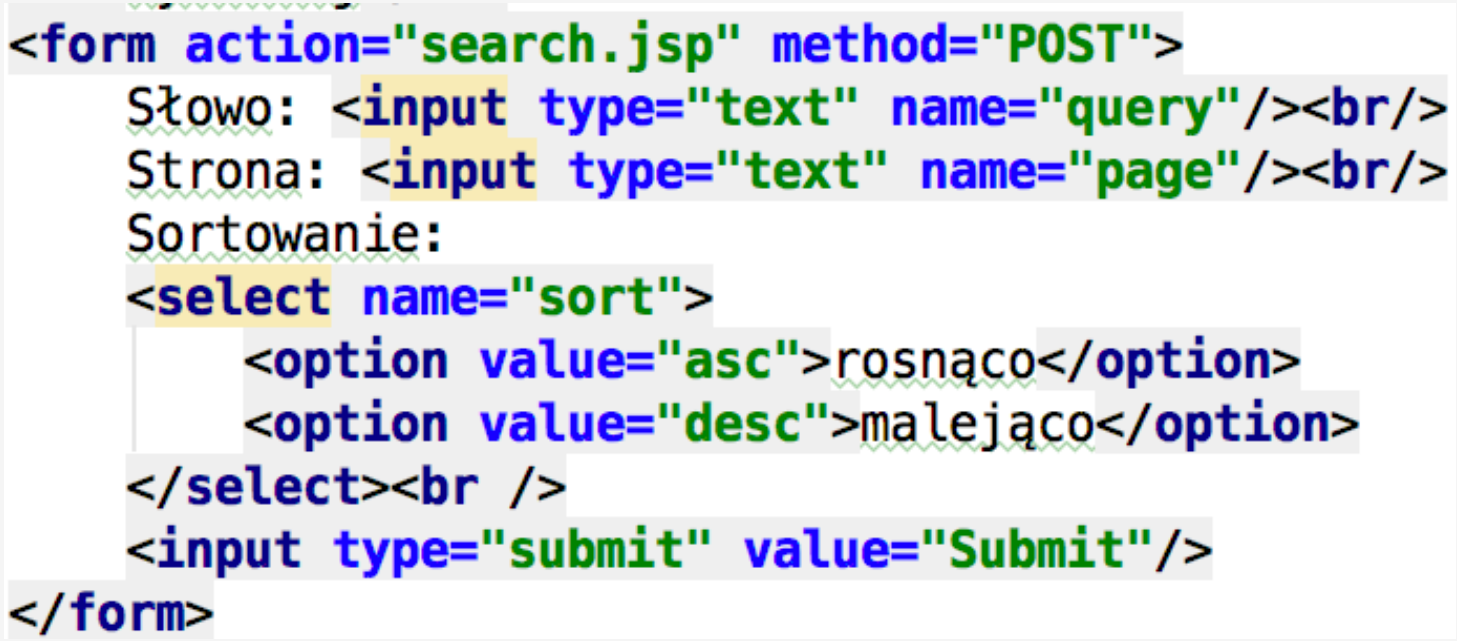
Ćwiczenia

Request GET

JSP - Metoda POST (1)

- Podobnie jak metoda POST służy do przesyłania danych z przeglądarki do serwera
- Aby wysłać dane metodą POST wykorzystuje się formularze HTML, gdzie atrybut `method` formularza ustawiamy na post
- Metody POST używa się, gdy odwołanie jest formą interakcji z użytkownikiem; operacja wywołuje zmiany na serwerze; operacja nie jest idempotentna; uploaduje się plik na serwerze
- Parametry przesyłane są w treści żądania, a nie po znaku ?
- Poszczególne parametry dostępne są za pomocą metod
  - `request.getParameter(„nazwaParametru”)`
  - `request.getParameterNames()`
  - `request.getParameterMap()`

JSP - Metoda POST (2)



```
<form action="search.jsp" method="POST">
    Słowo: <input type="text" name="query"/><br/>
    Strona: <input type="text" name="page"/><br/>
    Sortowanie:
    <select name="sort">
        <option value="asc">rosnąco</option>
        <option value="desc">malejąco</option>
    </select><br />
    <input type="submit" value="Submit"/>
</form>
```

Ćwiczenia

Request POST

JSP - Ciasteczka (1)

- Ciasteczka to małe tekstowe informacje trzymane po stronie klienta
- Działanie mechanizmu ciasteczek można podzielić na trzy fazy
  - serwer (w odpowiedzi) wysyła ciasteczko do przeglądarki klienta
  - przeglądarka zapisuje ciasteczko (na dysku)
  - podczas kolejnego żądania do serwera, przeglądarka odsyła ciasteczko wraz z żądaniem

```
Cookie cookie = new Cookie("searchId", String.valueOf(2138902773));
cookie.setMaxAge(60*60*24);
response.addCookie(cookie);
```

JSP - Ciasteczka (2)

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=C97D0F33B6CFFF25B0400618FAD89106; Path=/; HttpOnly
Set-Cookie: searchId=2138902773; Expires=Mon, 15-Feb-2016 14:54:00 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 450
Date: Sun, 14 Feb 2016 14:54:00 GMT
```

```
request.getCookies(); // ... and next get searchId cookie
%>CookieName: <b><%=cookie.getName()%></b>, CookieValue: <b><%=cookie.getValue()%></b><%
```

Szukane słowo: **jsp servlets**
Strona: **12**
Sortowanie: **rosnąco**

CookieName: **searchId**, CookieValue: **2138902773**

Ćwiczenia

Obsługa ciasteczek

JSP - Mechanizm sesji (1)

- Protokół HTTP jest bezstanowy
- Bez dodatkowych mechanizmów, serwer nie jest w stanie zidentyfikować (pamiętać) użytkownika podczas kilku jego żądań
- W celu identyfikacji użytkownika powstał mechanizm sesji
- W JSP realizuje go interfejs `HttpSession`
- Domyślnie strony JSP wspierają sesję
- Aby wyłączyć obsługę sesji na stronie należy użyć dyrektywy page: `<%@ page session=„false” %>`
- Identyfikator sesji jest nadawany przez server i ustawiany w ciasteczku o nazwie `JSESSIONID`

```
Set-Cookie: JSESSIONID=C97D0F33B6CFFF25B0400618FAD89106; Path=/; HttpOnly
```

JSP - Mechanizm sesji (2)

- **getId**() - zwraca identyfikator sesji
- **getCreationTime**() - zwraca timestamp utworzenia sesji
- **isNew**() - zwraca true, jeśli użytkownikowi została przydzielona właśnie sesja
- **getLastAccessedTime**() - zwraca timestamp ostatniego żądania użytkownika z zalogowaną sesją
- **getAttribute**(String name) - zwraca atrybut sesji o danej nazwie
- **setAttribute**(String name, Object value) - ustawia atrybut w sesji
- **removeAttribute**(String name) - usuwa atrybut z sesji
- **getMaxInactiveInterval**() - zwraca maksymalny czas utrzymywania sesji bez żądań użytkownika
- **invalidate**() - inwaliduje sesję i usuwa wszystkie obiekty z nią związane

Ćwiczenia

Obsługa sesji

Servlety

Servlety - Wprowadzenie

- Technologia Java Servlets (obok JSP) jest integralną częścią J2EE
- Umożliwia przetwarzanie żądań HTTP i generowanie odpowiedzi
- Servlety to klasy javy implementujące interfejs `HttpServlet`
- Dzięki nim (m.in.) separuje się warstwę logiki biznesowej od warstwy prezentacji
- Do uruchomienia servlety wymagają serwera webowego, np. Tomcata
- Konfigurację servletu można zawrzeć w adnotacji `@WebServlet` lub w pliku `web.xml`
- Na serwerze występuje zazwyczaj jedna instancja servletu, z której korzysta wiele wątków jednocześnie

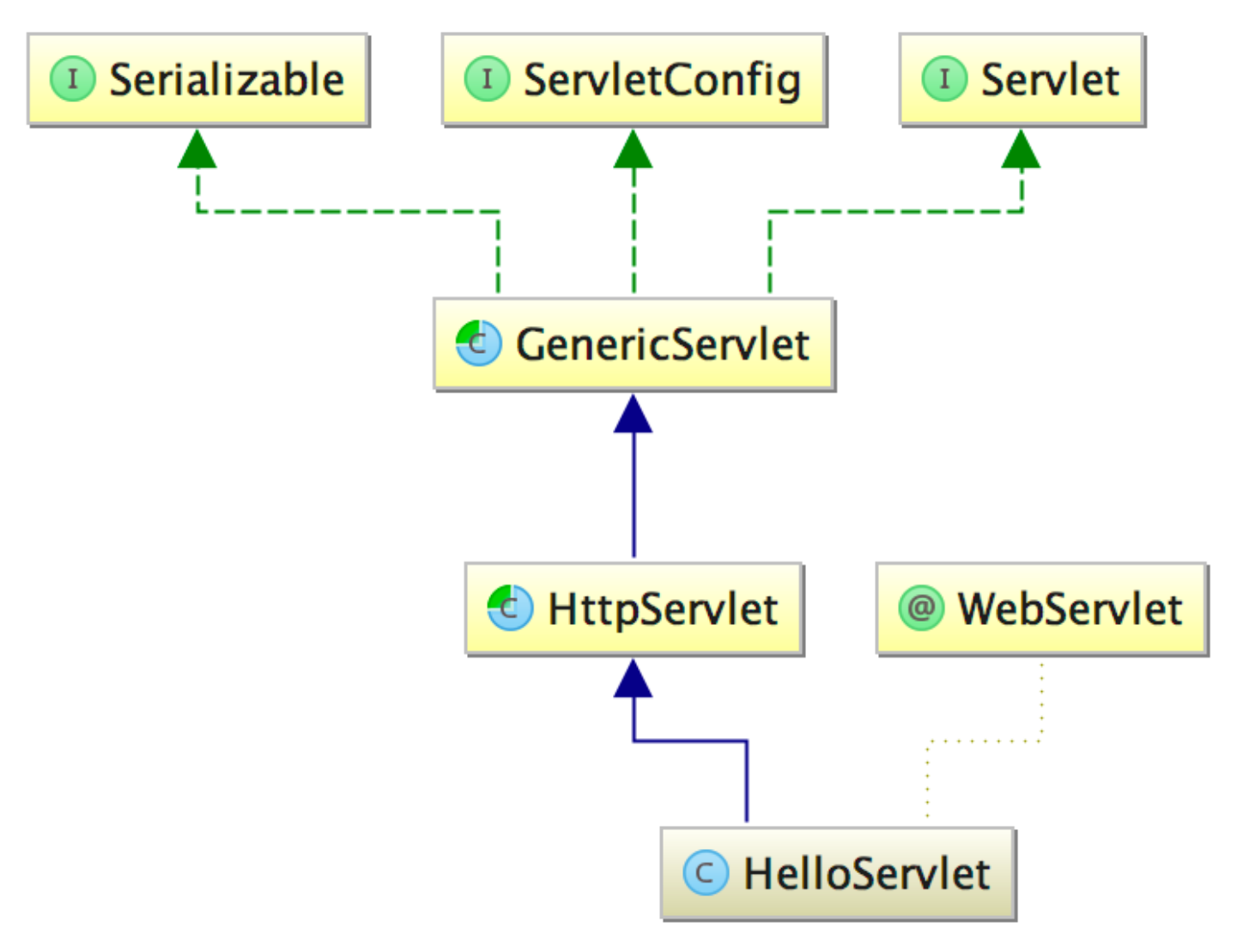
Servlety - Cykl życia servletu

- Podczas startu aplikacji lub pierwszego żądania, kontener web
  - ładuje klasę servletu
  - tworzy instancję klasy servletu
  - wywołuje metodę `init()` servletu
- Przetwarzanie żądań
  - wywoływanie metod obsługi żądań: metoda `service()` woła odpowiednią metodę `doGet()`, `doPost`, itd
- Zwalnianie zasobów
  - wywołanie metody `destroy()`

Servlety - Przetwarzanie żądania

1. Przeglądarka wysyła żądanie (*request*) HTTP do serwera
2. Serwer rozpoznaje żądanie do servletu (inicjalizuje go jeśli trzeba)
3. Żądanie klienta jest przetwarzane przez servlet
4. Servlet przygotowuje odpowiedź dla serwera
5. Odpowiedź przekazywana jest do serwera
6. Serwer odsyła odpowiedź (*response*) HTTP do przeglądarki

Servlety - Hello World - hierarchia

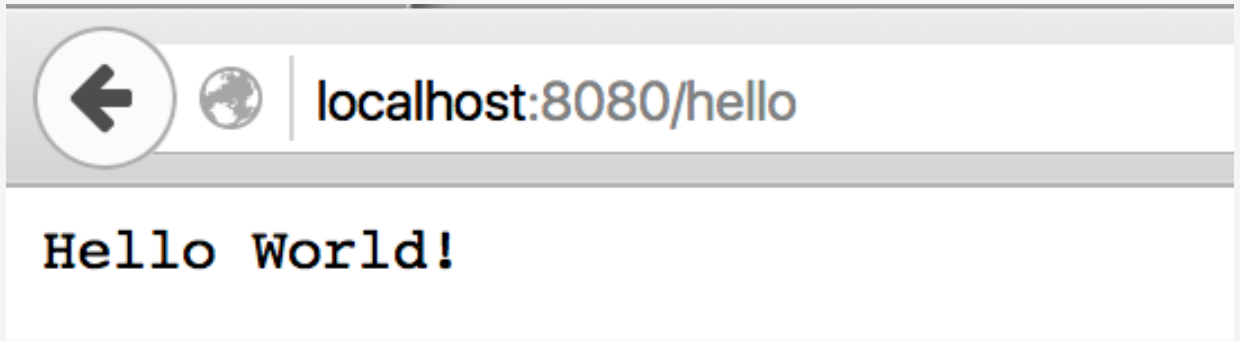


Servlety - Hello World - kod

```java
@WebServlet(name = "HelloServlet", value = "/hello")
public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("Hello World!");
    }
}
```



Servlety - Hello World - konfiguracja

```
@WebServlet(name = "HelloServlet", value = "/hello")
```

lub

```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>foo.bar.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

Ćwiczenie

Hello World!

Servlety - Konfiguracja - adnotacje

```
@WebServlet(
        name = "HelloServlet",
        urlPatterns = {"/hello", "/Hello"},
        initParams = {
                @WebInitParam(name = "who", value = "World"),
                @WebInitParam(name = "times", value = "5")
        },
        loadOnStartup = 1)
public class HelloServlet extends HttpServlet {
```

Servlety - Konfiguracja - web.xml

```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>foo.bar.HelloServlet</servlet-class>
    <init-param>
        <param-name>who</param-name>
        <param-value>World</param-value>
    </init-param>
    <init-param>
        <param-name>times</param-name>
        <param-value>5</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
    <url-pattern>/Hello</url-pattern>
</servlet-mapping>
```

Ćwiczenia

Servlety, konfiguracja, parametryzacja

Servlety - Filtry

- Filtry przekształcają zawartość żądania i/lub odpowiedzi
- Są klasami javy implementującymi interfejs `Filter`
- Dzięki nim oddziela się powtarzalną logikę od specyficznej logiki konkretnego zasobu
- Umożliwiają wprowadzenie aspektu do specyficznej logiki

- Są realizacją wzorca projektowego `Decorator`
- Przetwarzanie/dopasowywanie filtrów jest natomiast realizacją wzorca projektowego `Chain of Responsibility`
- Konfigurację filtra można zawrzeć w adnotacji `@Filter` lub w pliku `web.xml`
- Przykładowe użycia filtrów, to:
  - walidacja
  - kompresja
  - weryfikacja

Servlety - Filtry - cykl życia

- Podczas pierwszego żądania, kontener web
  - ładuje klasę filtra
  - tworzy instancję klasy filtra
  - wywołuje metodę `init()` filtra
- Przetwarzanie żądań
  - jeśli filtr podłączony jest pod żądany servlet, wykonywana jest logika filtra zawarta w metodzie `doFilter()`
- Zwalnianie zasobów
  - wywołanie metody `destroy()` filtra

Servlety - Filtry - implementacja

```
@Override
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
        throws ServletException, IOException {

    // logika wykonywana przed logika servletu (opcjonalna)

    // przekazanie sterowania do kolejnego filtra lub servletu
    chain.doFilter(req, resp);

    // logika wykonywana po wykonaniu logiki servletu (opcjonalna)
}
```

Servlety - Filtry - konfiguracja

```java
//@WebFilter(filterName = "HelloFilter", urlPatterns = {"/hello"})
@WebFilter(filterName = "HelloFilter", servletNames = {"HelloServlet"})
public class HelloFilter implements Filter {
```

```xml
<filter>
    <filter-name>HelloFilter</filter-name>
    <filter-class>foo.bar.HelloFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>HelloFilter</filter-name>
    <!--<url-pattern>/hello</url-pattern>-->
    <servlet-name>HelloServlet</servlet-name>
</filter-mapping>
```

Ćwiczenia

Filtry, konfiguracja, parametryzacja

Servlety - Przekierowania (po stronie klienta)

```
response.sendRedirect("/index.jsp");
```

```
iMac:~ rafos$ curl -v "http://localhost:8080/hello"
*   Trying ::1...
* Connected to localhost (::1) port 8080 (#0)
> GET /hello HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 302 Found
< Server: Apache-Coyote/1.1
< Location: /index.jsp
< Content-Length: 0
< Date: Tue, 16 Feb 2016 21:47:06 GMT
<
* Connection #0 to host localhost left intact
```

Servlety - Przekierowania (po stronie servera)

Przekierowanie na stronę

```
RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp");
dispatcher.forward(request, response);
```

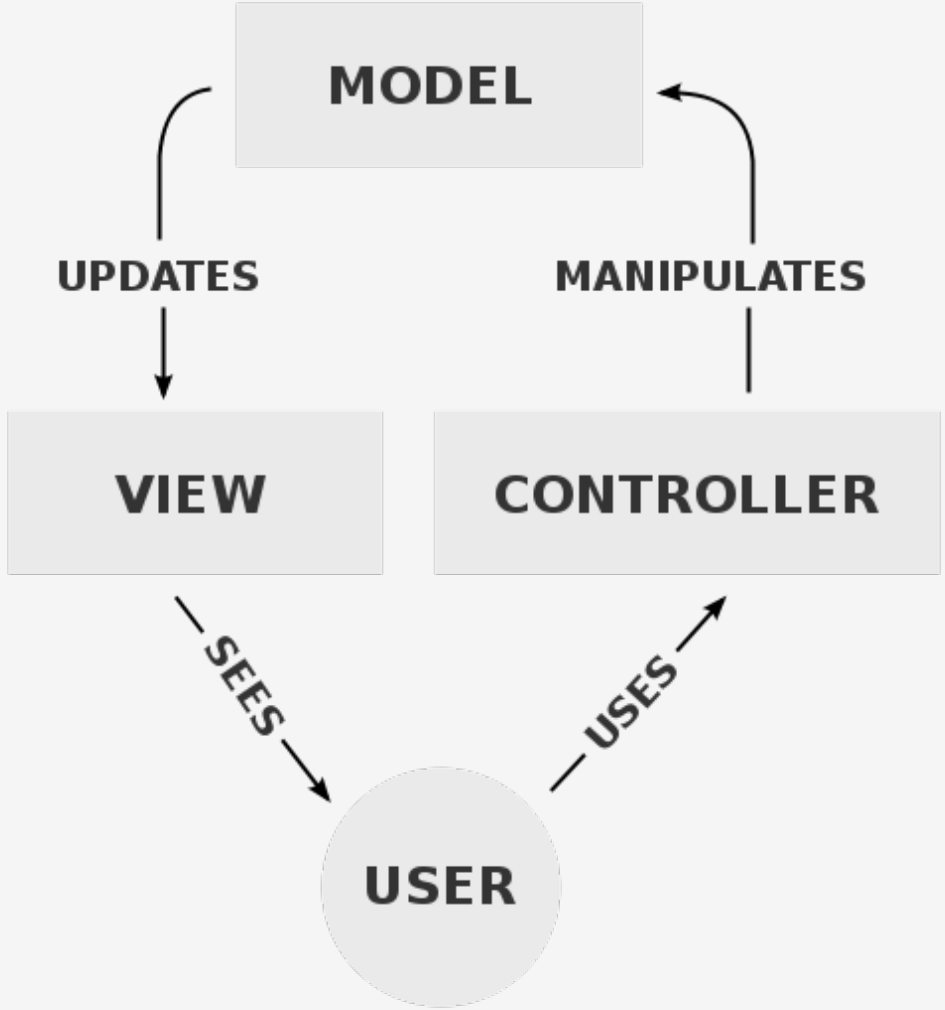
Dołączenie strony

```
RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp");
dispatcher.include(request, response);
```

Ćwiczenia

Przekierowania i dołączania

Servlety - MVC - diagram



https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/500px-MVC-Process.svg.png

Servlety - MVC

- Istotą wzorca MVC (Model View Controller) jest oddzielenie logiki biznesowej (modelu) od prezentacji (view) oraz umieszczenie pomiędzy nimi sterowania (kontrolera)
- Separacja odpowiedzialności umożliwia łatwiejsze zmiany w poszczególnych warstwach, np. zmianę prezentacji, dodanie innych form (czytników) prezentacji
- **Model** - przechowuje logikę biznesową oraz stan aplikacji. Zna reguły uzyskiwania i aktualizacji tego stanu
- **View** - odpowiada za prezentację. Otrzymuje od kontrolera stan modelu
- **Controller** - pobiera żądania użytkownika i określa, co te dane znaczą dla modelu. Nakazuje modelowi aktualizację i sprawia, że (nowy) stan modelu jest dostępny dla widoku

Servlety - JDBC (1)

- Zestaw interfejsów pozwalających na komunikację z bazami danych
- **Connection**
  - reprezentuje połączenie z bazą danych
  - umożliwia tworzenie obiektów do obsługi zapytań
- **Statement**
  - służy do wykonywania statycznych zapytań
- **PreparedStatement**
  - służy do wykonywania dynamicznych (bindowanie zmiennych) zapytań
  - pozwala na szybsze wykonywania podobnych zapytań (prekompilacja)
- **ResultSet**
  - służy do przeglądania wyników
  - istnieje dokładnie jeden dla jednego obiektu Statement lub PreperedStatement

Servlety - JDBC (2)

```
User user = new User();

Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/rafos", "user_test", "test01");

PreparedStatement preparedStatement = null;
ResultSet resultSet = null;
```

```
try {
    String query = "SELECT id, login, nickName FROM user WHERE id=?";
    preparedStatement = conn.prepareStatement(query);
    preparedStatement.setInt(1, id);
    resultSet = preparedStatement.executeQuery();

    while (resultSet.next()) {
        user.setId(resultSet.getInt("id"));
        user.setLogin(resultSet.getString("login"));
        user.setNickName(resultSet.getString("nickName"));
    }

} catch (SQLException e) {
```

Servlety - JDBC (3)

```
} finally {
    if (resultSet != null) {
        try {
            resultSet.close();
        } catch (SQLException e) {
            log.error(e.getMessage(), e);
        }
    }
    if (preparedStatement != null) {
        try {
            preparedStatement.close();
        } catch (SQLException e) {
            log.error(e.getMessage(), e);
        }
    }
    try {
        if (conn != null && !conn.isClosed()) {
            conn.close();
        }
    } catch (SQLException e) {
        log.error(e.getMessage(), e);
    }
}
return user;
```

Projekt

Twitter - prosty serwis internetowy umożliwiający dodawanie krótkich wiadomości tekstowych

Dodatkowe informacje

https://docs.oracle.com/javaee/5/tutorial/doc/javaeetutorial5.pdf

http://download.oracle.com/otn-pub/jcp/servlet-3_1-fr-eval-spec/servlet-3_1-final.pdf

http://download.oracle.com/otn-pub/jcp/jsp-2.1-fr-eval-spec-oth-JSpec/jsp-2_1-fr-spec.pdf

http://download.oracle.com/otn-pub/jcp/jsp-2.1-fr-eval-spec-oth-JSpec/jsp-2_1-fr-spec-el.pdf