

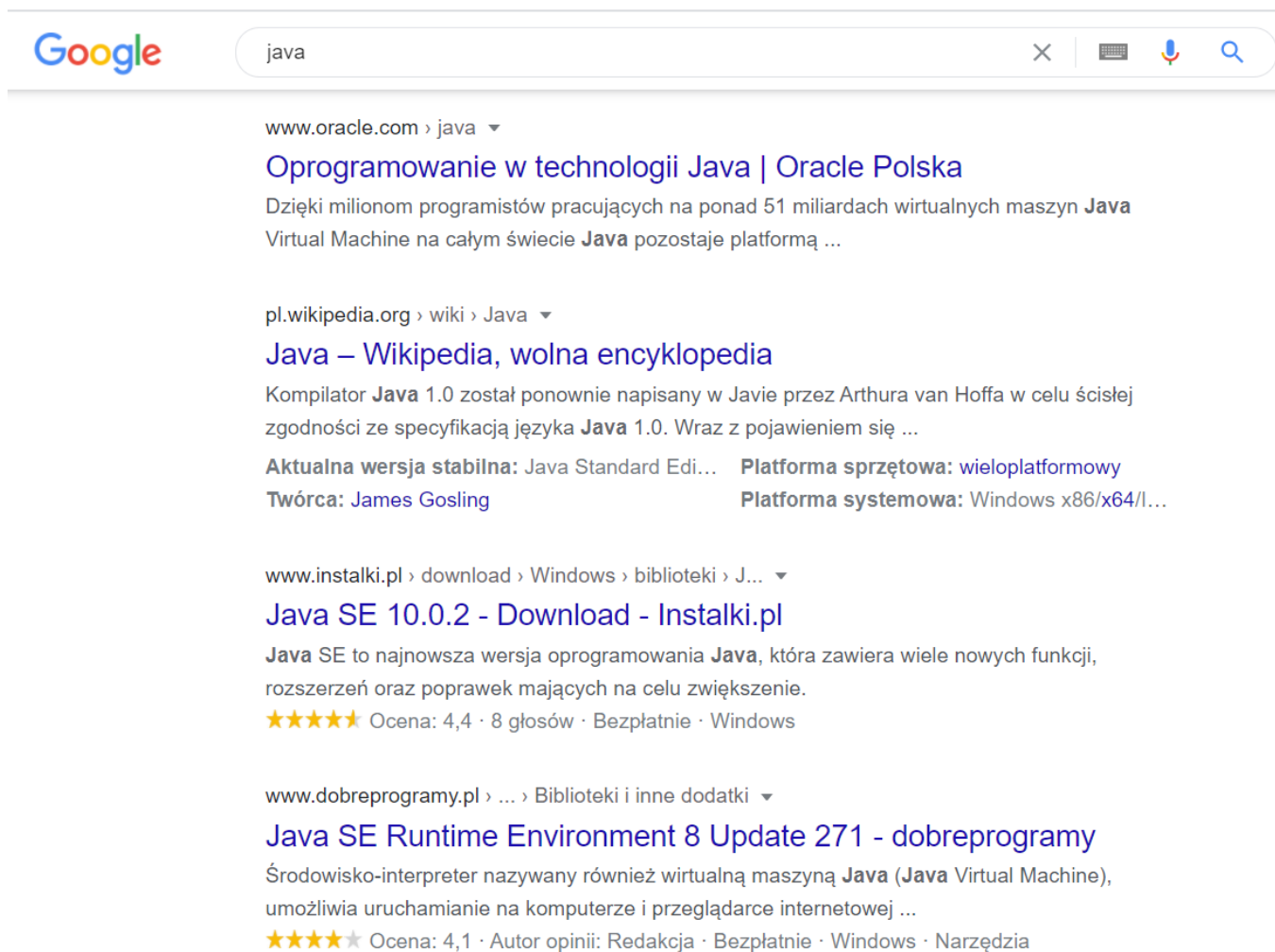


**Wyższa Szkoła Bankowa
w Gdańsku**

Java Server Pages

Katarzyna Duchowska

JSP – jaki problem chcemy rozwiązać?



The image shows a Google search interface with the query 'java'. The results are filtered by location to Poland. The first result is from Oracle Polska, describing Java as a platform. The second is a Wikipedia entry for Java. The third is from Instalki.pl, advertising Java SE 10.0.2. The fourth is from dobreprogramy.pl, advertising Java SE Runtime Environment 8 Update 271.

Google

java

www.oracle.com › java ▾

[Oprogramowanie w technologii Java | Oracle Polska](#)

Dzięki milionom programistów pracujących na ponad 51 miliardach wirtualnych maszyn **Java** Virtual Machine na całym świecie **Java** pozostaje platformą ...

pl.wikipedia.org › wiki › Java ▾

[Java – Wikipedia, wolna encyklopedia](#)

Kompilator **Java** 1.0 został ponownie napisany w Javie przez Arthura van Hoffa w celu ścisłej zgodności ze specyfikacją języka **Java** 1.0. Wraz z pojawieniem się ...

Aktualna wersja stabilna: Java Standard Edi... Platforma sprzętowa: wieloplatformowy

Twórca: James Gosling Platforma systemowa: Windows x86/x64/I...

www.instalki.pl › download › Windows › biblioteki › J... ▾

[Java SE 10.0.2 - Download - Instalki.pl](#)

Java SE to najnowsza wersja oprogramowania **Java**, która zawiera wiele nowych funkcji, rozszerzeń oraz poprawek mających na celu zwiększenie.

★★★★★ Ocena: 4,4 · 8 głosów · Bezpłatnie · Windows

www.dobreprogramy.pl › ... › Biblioteki i inne dodatki ▾

[Java SE Runtime Environment 8 Update 271 - dobreprogramy](#)

Środowisko-interpreter nazywany również wirtualną maszyną **Java** (**Java** Virtual Machine), umożliwia uruchamianie na komputerze i przeglądarce internetowej ...

★★★★★ Ocena: 4,1 · Autor opinii: Redakcja · Bezpłatnie · Windows · Narzędzia

JSP – jaki problem chcemy rozwiązać?

```
><div class="g" data-hveid="CAMQAA">...</div>
><div class="g">...</div>
▼<div class="g">
  <!--m-->
  ▼<div class="rc" data-hveid="CAEQAA" data-ved="2ahUKEwjmg8artIDuAhUul4sKHYgZD2QQF5gAMAR6BAGBEAA">
    ><div class="yuRUBf">...</div>
    ><div class="IsZvec">...</div>
    ><div jscontroller="m6a0l" id="eob_10" jsdata="fxg5tf;;BsY4KI" jsaction="rcuQ6b:npT2md" data-ved="2ahUKEwjmg8artI
    </div>
    <!--n-->
  </div>
  <span id="fld"></span>
  <script nonce="4FCld30/GmCv14/EHWKfgg==">(function(){
    var a=google.time();google.tick("load","aft",a);}).call(this);</script>
  ><script nonce="4FCld30/GmCv14/EHWKfgg==">...</script>
  ><style>...</style>
  ▼<div class="g">
    <!--m-->
    ▼<div class="rc" data-hveid="CAsQAA" data-ved="2ahUKEwjmg8artIDuAhUul4sKHYgZD2QQF5gAMAV6BAGLEAA"> == $0
      ><div class="yuRUBf">...</div>
      ><div class="IsZvec">...</div>
      ><div jscontroller="m6a0l" id="eob_25" jsdata="fxg5tf;;BsY4K4" jsaction="rcuQ6b:npT2md" data-ved="2ahUKEwjmg8artI
      </div>
      <!--n-->
    </div>
  ><div class="g">...</div>
  ><div class="g">...</div>
  ><div class="g">...</div>
  ><div class="g">...</div>
  ><div class="g">...</div>
  ...
```

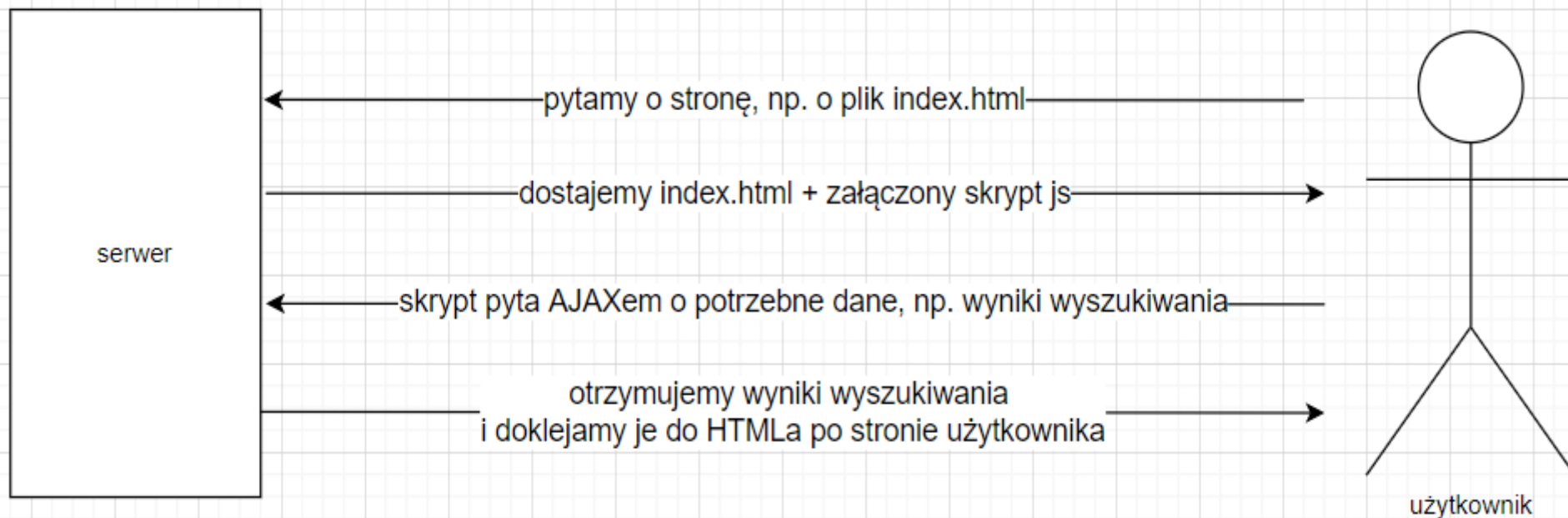
JSP – jaki problem chcemy rozwiązać?


- Często na stronach pojawiają *podobne struktury*
- Przykład na poprzednim slajdzie – lista wyników wyszukiwania Google
- Każdy wynik ma podobną strukturę – link w nagłówku, krótki opis, czasem ocenę lub inne informacje
- W jaki sposób przygotować HTML, żeby generował znaczniki w pętli lub dodawał niektóre węzły warunkowo?

Jakie rozwiązanie już znamy?

- Dodawanie/usuwanie węzłów poprzez skrypty JavaScript
- Czyli np. pytamy o dane AJAXem, a odpowiedź doklejamy do strony korzystając z kodu JS – możemy łatwo napisać pętle po wynikach lub jakieś instrukcje warunkowe w JS
- Czasem jednak wygodniej byłoby wykonać taką pętle po stronie serwera i wysłać użytkownikowi już gotową stronę...

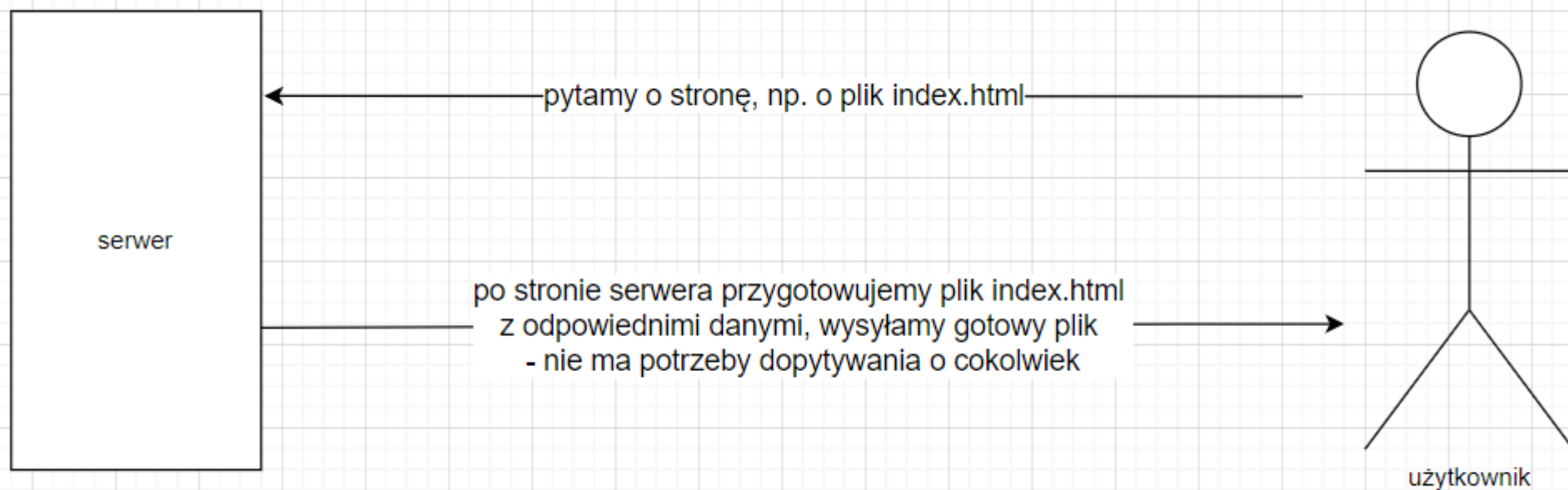
Przykład 1 – dopytujemy o dane AJAXem





Czy dałoby się prościej? Skoro użytkownik i tak chce zobaczyć wyniki wyszukiwania, może nie warto o nie dopytywać, a lepiej wysłać od razu?

Przykład 2 – przygotowujemy całego HTMLa po stronie serwera



Kiedy AJAX, a kiedy przygotowywać HTML po stronie serwera?

- Na tym kursie skupiamy się na backendzie, więc *raczej* HTMLe będziemy przygotowywać po stronie serwera ;)
- AJAX może być przydatny dla długotrwałych operacji lub niewielkich zmian w strukturze HTMLa
- Przy zmianie strony *raczej* będzie lepiej przygotować HTML po stronie serwera
- Nie mówimy o sytuacji, gdy front jest osobną aplikacją (np. Angular, React) – takie przypadki będziemy sobie jeszcze omawiać na innych przedmiotach

Jak dodać instrukcje sterujące do HTML?

- JSP – Java Server Pages
- Za Wikipedią:

JSP (*ang. JavaServer Pages*) – technologia umożliwiająca tworzenie dynamicznych dokumentów *WWW* w formatach *HTML*, *XHTML*, *DHTML* oraz *XML* z wykorzystaniem języka *Java*, wpleczonego w kod *HTML*.

W jaki sposób przygotować projekt, który wykorzystuje JSP?

- Np. tworząc aplikację w Spring Boocie
- Instrukcja jest w prezentacji **JSP – konfiguracja projektu**
- My do ćwiczeń skorzystamy z przygotowanego już projektu na GitHubie
- Ale najpierw trochę teorii :)

Java Server Pages

- Kod będziemy pisać w plikach z rozszerzeniem *.jsp*
- Rozszerzenie to nie wymusza żadnej szczególnej składni (innej niż HTML) – możemy w tym pliku pisać czysty HTML i wszystko zadziała
- Możemy jednak także używać w tych plikach języka Java
- Wyrażenia w języku Java będą obliczane po stronie serwera, do przeglądarki przyjdzie już gotowy HTML – nie będzie śladu, że coś było tam liczone
- Jak to zrobić?

JSP - wyrażenia

- Gdy chcemy wyświetlić na stronie lub przypisać jakiemuś atrybutowi w dokumencie pewną wartość, korzystamy z tzw. *expressions*
- Można to traktować jako coś podobnego do „wypisania na konsolę”, tylko że naszą konsolą będzie dokument HTML
- Składnia: **<%= wyrażenie %>**

JSP - wyrażenia

```
<h3>EXPRESSIONS</h3>
<div>
  Dzisiejsza data to <%= new Date() %>
</div>
```

EXPRESSIONS

Dzisiejsza data to Sun Jan 03 20:20:58 CET 2021

JSP - deklaracje

- Poprzez deklarację możemy zdefiniować jakąś zmienną lub funkcję
- Składnia: `<%! kod %>`

JSP - deklaracje

```
<section>
  <h3>DECLARATIONS</h3>
  <%!
    String makePokemon(String text) {
      StringBuilder result = new StringBuilder();
      String[] stringArray = text.split("");
      for (int i = 0; i < stringArray.length; i++) {
        result.append(i % 2 == 0 ? stringArray[i].toLowerCase() : stringArray[i].toUpperCase());
      }
      return result.toString();
    }
  %>
  <div>
    Zmieniamy w pokemony: <br/>
    <%=makePokemon("JSP jest mega super fajne")%> <br/>
    <%=makePokemon("JAKI PIĘKNY DZIOŃEK DZISIAJ!")%>
  </div>
</section>
```

DECLARATIONS

Zmieniamy w pokemony:
jSp jEsT MeGa sUpEr fAjNe
jAkI PiĘkNy dZiOnEk dZiSiAj!

JSP - skrypty

- Kiedy chcemy wykonać jakieś instrukcje
- Można je mieszać z kodem HTML!
- Składnia: **<% jedna lub więcej linii kodu %>**

JSP - skrypty

```
<section>
  <h3>SCRIPTETS</h3>
  <% boolean showSpan = true;
    if (showSpan) { %>

    <span>WIDZIMY SPAN</span>

  <% } else { %>

    <span>NIE WIDZIMY SPANA</span>

  <% } %>
</section>
```

SCRIPTETS

WIDZIMY SPAN

JSP - skrypty

```
<section>
  <h3>SCRIPTLETS - LOOP</h3>
  <% for (int i = 0; i < 10; i++) { %>

    <span><%= i %></span>

    <% } %>
</section>
```

SCRIPTLETS - LOOP

0 1 2 3 4 5 6 7 8 9

JSP - dyrektywy

- Składnia: `<%@ wyrażenie %>`

- Przykłady:

- Określenie kodowania znaków:

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

- Import klasy, którą wykorzystujemy w pliku `.jsp`:

```
<%@ page import="java.util.Date" %>
<html>
<head>
  <title>TEST JSP</title>
</head>
<body>
  <section>
    <h3>EXPRESSIONS</h3>
    <div>
      Dzisiejsza data to <%= new Date() %>
    </div>
  </section>
</body>
</html>
```

JSP – fajnie, ale...

- Czy to czytelne mieszać kod HTML z kodem Javy?
- Czy to mądre pisać logikę w plikach, w których jest HTML?
- Raczej nie :)
- Co więc robić?



JSP – fajnie, ale...

- Niektórych pętli czy instrukcji warunkowych nie unikniemy
- Czasem będzie też trzeba wywołać jakąś funkcję

Sugestia 1

- *Raczej nie deklarujemy funkcji w plikach .jsp - lepiej mieć je w jakiejś klasie javowej:*

```
<section>
  <h3>Przeniesienie funkcji do osobnego pliku</h3>
  <div>
    Zmieniamy w pokemony: <br/>
    <%=StringUtil.makePokemon("JSP jest mega super fajne")%> <br/>
    <%=StringUtil.makePokemon("JAKI PIEKNY DZIOONEK DZISIAJ!")%>
  </div>
</section>
```

```
public class StringUtil {

    public static String makePokemon(String text) {
        StringBuilder result = new StringBuilder();
        String[] stringArray = text.split( regex: "" );
        for (int i = 0; i < stringArray.length; i++) {
            result.append(i % 2 == 0 ? stringArray[i].toLowerCase() : stringArray[i].toUpperCase());
        }
        return result.toString();
    }
}
```

Sugestia 2

- Do instrukcji warunkowych czytelniejsze *wyduje się* użycie JSTL

```
<section>
  <h3>SCRIPTLETS</h3>
  <% boolean showSpan = true;
    if (showSpan) { %>

    <span>WIDZIMY SPAN</span>

  <% } else { %>

  <span>NIE WIDZIMY SPANA</span>

  <% } %>

  <h3>SCRIPTLETS - LOOP</h3>
  <% for (int i = 0; i < 10; i++) { %>

  <span><%= i %></span>

  <% } %>
</section>
```

```
<section>
  <h3>JSTL</h3>
  <c:set var="showSpanJSTL" value="${true}"/>
  <c:choose>
    <c:when test="${showSpanJSTL}">
      <span>WIDZIMY SPAN</span>
    </c:when>
    <c:otherwise>
      <span>NIE WIDZIMY SPANA</span>
    </c:otherwise>
  </c:choose>

  <h3>JSTL - LOOP</h3>
  <c:forEach var="i" begin="0" end="9">
    <span>${i}</span>
  </c:forEach>
</section>
```


JSTL – JSP Standard Tag Library

- Tagi, które umożliwiają skorzystanie z podstawowych funkcji JSP, zwykle wykorzystywanych w aplikacjach
- Głównie – instrukcje warunkowe, pętle, deklaracje zmiennych itp.
- Przykłady – w kodzie:
https://github.com/Kaatarzyna/WSB_JSP

JSTL - konfiguracja

- Dodanie wpisu w *pom.xml* → patrz prezentacja „JSP – konfiguracja projektu”
- Import (np. dla *Core*):

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

- W miarę spoko opisane przykłady:
https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm

JSP EL – JSP Expression Language

- Umożliwia dynamiczny dostęp do obiektów i funkcji wewnątrz JSP
- Składnia: `${..}`
- Nie każdy kod Javowy zadziała wewnątrz EL – można korzystać ze zdefiniowanych już obiektów, wykonywać metody statyczne, korzystać z integerów, floatów, stringów, booleanów, podstawowych operatorów (arytmetycznych i logicznych)
- Nie da się np. stworzyć w EL nowego obiektu

Przekazywanie modelu do widoku

- ModelAndView (starsze podejście):

```
@RequestMapping("/countries")
public ModelAndView countries() {
    ModelAndView modelAndView = new ModelAndView( viewName: "countries/index");
    modelAndView.addObject( attributeName: "countries", countryRepository.findAll());
    return modelAndView;
}
```

Przekazywanie modelu do widoku

- Model (nowsze podejście):

```
@RequestMapping("/countriesModel")
public String countriesModel(Model model) {
    model.addAttribute("countries", countryRepository.findAll());
    return "countries/index";
}
```

Przekazywanie modelu do widoku

- Czego używać?

<https://stackoverflow.com/questions/16951609/when-to-use-modelandview-vs-model-in-spring>

3 Answers

Active

Oldest

Votes



52



I always use the approach where controller methods return `ModelAndView`. Simply because it tends to make the controller methods a little more terse. The method parameters are now strictly `input` parameters. And all `output` related data is contained in the object returned from the method.

The `ModelAndView` style seems to resonate with people who don't like updating input parameters to a method. Sticking to the belief that this would constitute a side effect, a dangerous pattern because you cannot reliably predict what the method is going to do - It could return data in the returned object, or it could have updated anything in any of the input arguments.

So some people will still continue to prefer `ModelAndView`.

The new style with `Model` as method parameter and returned string as view name. Seems to have come from a slightly different design approach. Here the model objects are considered to be sort of events or items that are passed to multiple handlers, before being returned to the view where they are rendered. It reminds me how events are handled in the AWT / Swing world. This model is more coherent with the approach where multiple handlers could build on top of the `Model` objects, till it reaches a view.

So at the end of the day, there does not seem to be a definite reason to criticise or promote either approach. You should use the style that feels more consistent to your overall design philosophy.

Hope this helps.

Obiekty „wbudowane”

- Mamy w JSP też dostęp do pewnych obiektów „wbudowanych”:
 - ***request*** - żądanie http (nagłówki, dane formularzy) - [HttpServletRequest](#)
 - ***response*** - wsparcie dla wysyłania odpowiedzi - [HttpServletResponse](#)
 - ***out*** - "wypisuje treści" do htmla - [JspWriter](#)
 - ***session*** - unikalna sesja dla każdego użytkownika aplikacji - [HttpSession](#)
 - ***application*** - współdzielone dane dla wszystkich użytkowników aplikacji - [ServletContext](#)



Ćwiczenia :)

- Plik *exercise.md* w https://github.com/Kaatarzyna/WSB_JSP