Hello!

Rafał Misiak

Java Developer dla Stibo Systems (DK) w Ciklum

slack: @rafalmisiak

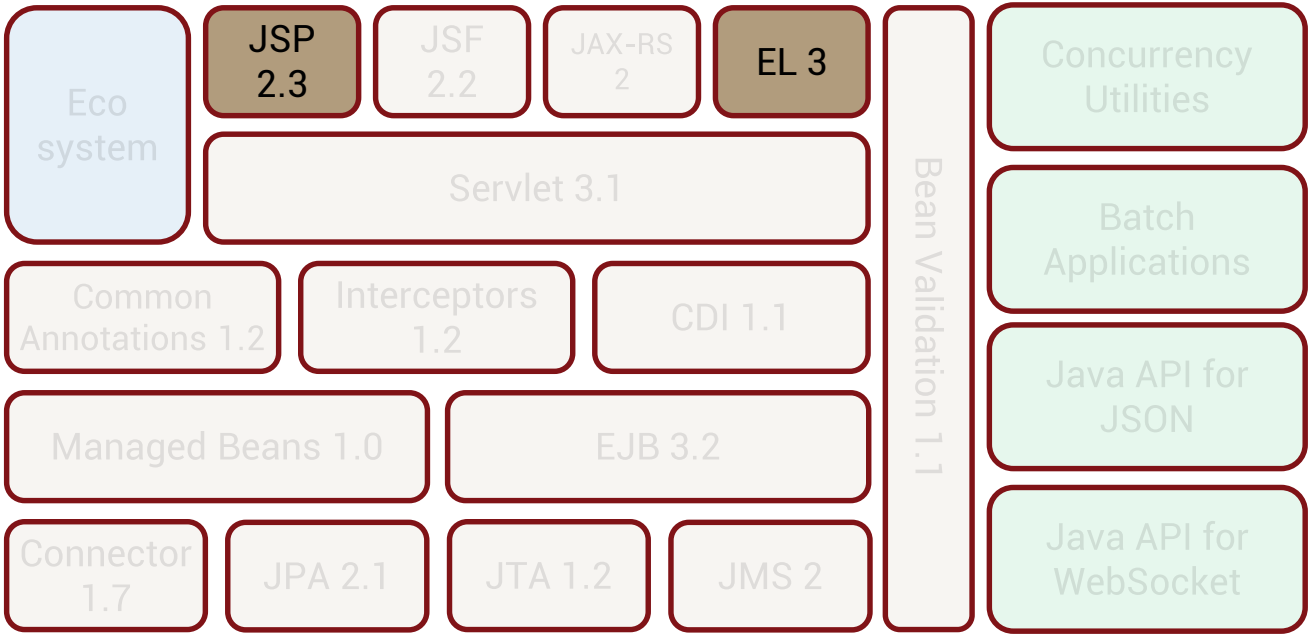
rafalmisiak@gmail.com

JEE
Techniki Zaawansowane

1.
JSP

Java Server Pages

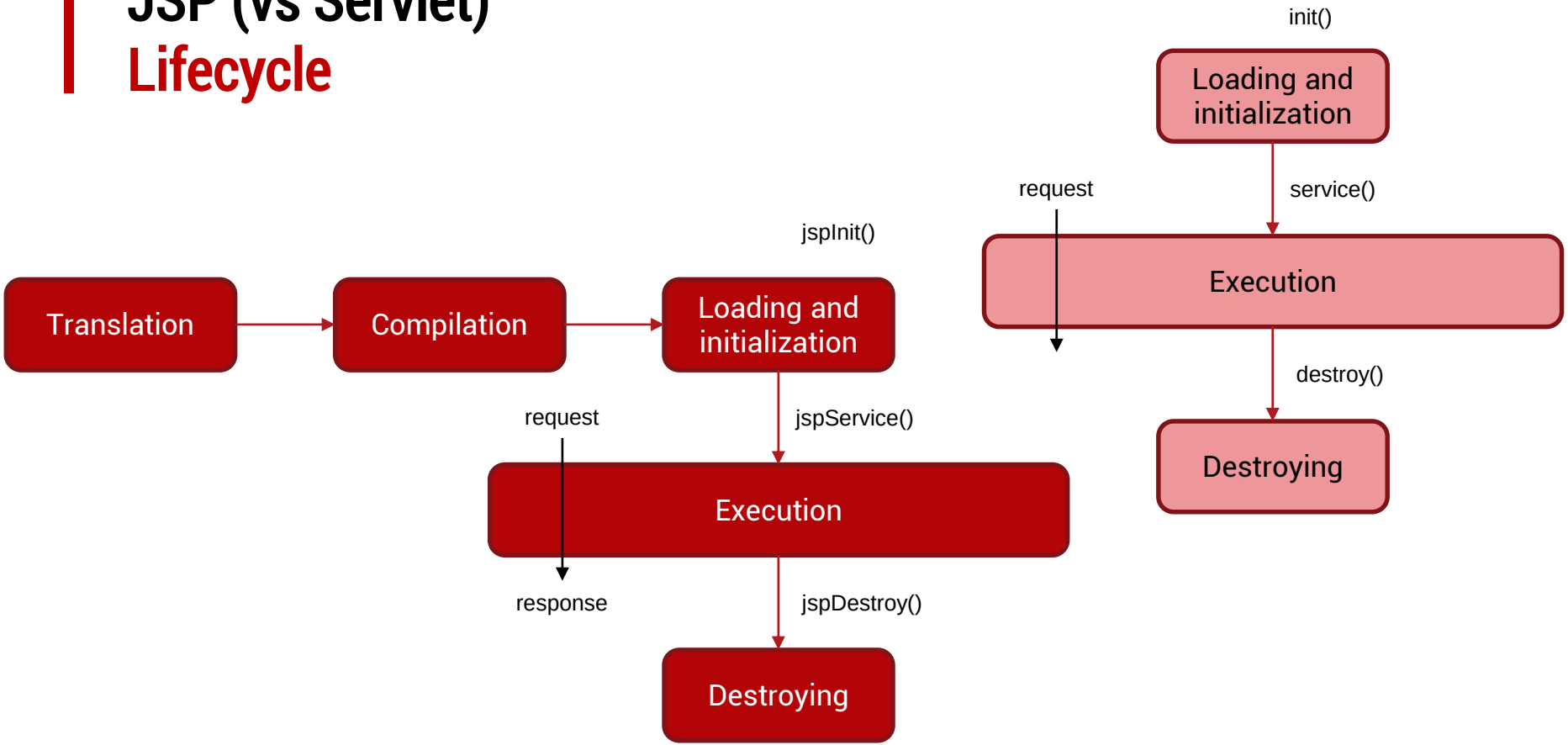
JEE 7


Eco system
JSP 2.3
JSF 2.2
JAX-RS 2
EL 3
Servlet 3.1
Common Annotations 1.2
Interceptors 1.2
CDI 1.1
Managed Beans 1.0
EJB 3.2
Connector 1.7
JPA 2.1
JTA 1.2
JMS 2
Bean Validation 1.1
Concurrency Utilities
Batch Applications
Java API for JSON
Java API for WebSocket

Servlets vs JSP

- JSP ułatwia tworzenie dynamicznych stron
- Umożliwia w sposób wygodny tworzenie kodu HTML/JS/CSS
- Kontener webowy kompiluje JSP do servletu, a następnie wykonuje go jak każdy inny

JSP (vs Servlet)
Lifecycle



JSP
lifecycle

- **Translation** – kontener webowy tłumaczy kod JSP na kod Java. Jest kod odwzorowujący servlet
- **Compilation** – kontener webowy kompiluje kod źródłowy, tworząc pliki class.
- **Loading and initialization** – kontener ładuje skompilowany servlet i tworzy jego instanację. Zaimplementowana przez kontener metoda init jest wykonywana
- **Execution** – zaimplementowana przez kontener metoda serwisowa jest wywoływana na każde żądanie

Zadanie 1.1
Pierwsza strona JSP

Znajdź w strukturze swojego projektu przykładową stronę index.jsp będącą stroną powitalną. Nie istnieje? Stwórz ją!

Przekształć jej kontent w treść powitalną swojego narzędzia wyszukiwania i zarządzania użytkownikami. Na przykład umieść tam swoje inicjały.

Przygotuj menu, którego linki będziesz uzupełniać w kolejnych zadaniach. Menu powinno zawierać odnośniki do stron: dodaj użytkownika, lista użytkowników.

Server default
welcome page

Kontener webowy domyślnie szuka pliku index.jsp.

Chcąc podać listę lub zmienić tę konfigurację, do pliku web.xml dodajemy odpowiednie informacje:

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
```

Zadanie 1.2
welcome page

Przeprowadź eksperyment: dodaj nowy plik o nazwie index.html, wprowadzając łatwy do zidentyfikowania tekst. Zmodyfikuj web.xml definiując listę welcome files jak przedstawiono na poprzednim slajdzie. Wykonaj deploy aplikacji.

Odwiedź adres: http://127.0.0.1:8080/

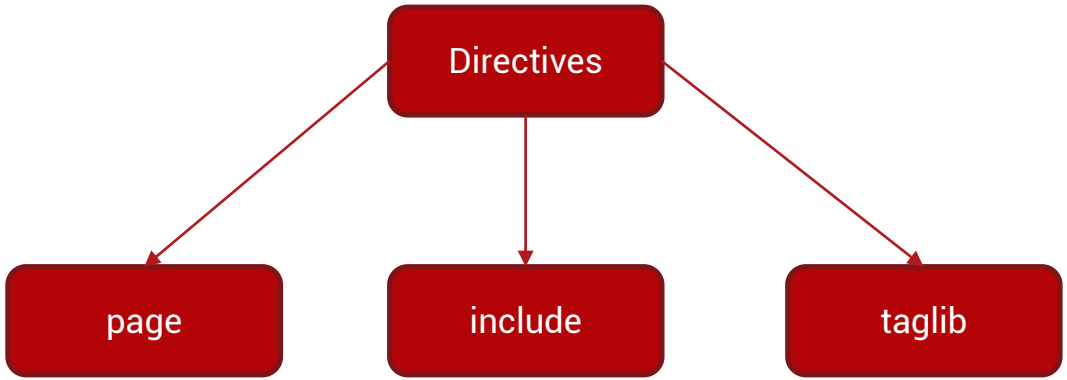
Wyciągnij wnioski i skasuj index.html.

2.
JSP: Directives

Sterowanie translacją stron

Directives
syntax

<%@ directive attribute="value" %>



page
directive

Dyrektywa page używana jest w pierwszym kroku, translacji JSP. Można jej użyć w dowolnym miejscu strony, jednak konwencja i dobre praktyki nakazują używania jej w górnej części strony.

```
<%@ page import="java.util.Date" %>
```

page directive
attributes

contentType – definiuje contentype dla strony, domyślnie: **text/html**
extends – definiuje nazwę servletu, który będzie rozszerzany przez servlet wygenerowany z naszej strony jsp
errorPage – adres pod który zostanie przekierowany request w przypadku wystąpienia RuntimeException
import – lista klas oddzielonych przecinkami, których import jest wymagany do poprawnego działania strony
info – zawiera informacje jakie zostaną zwrócone po wykonaniu metody getServletInfo()
isErrorPage – przyjmuje wartości true lub **false** i opisuje stronę jako stronę błędów
pageEncoding – definiuje kodowanie strony

include
directive

Dyrektywa include używana jest w pierwszym kroku, translacji JSP. Można jej użyć w dowolnym miejscu strony i odpowiedzialna jest za łączenie stron w jedną.

<%@ include file=„search.jsp" %>

taglib
directive

Dyrektywy można użyć w dowolnym miejscu strony. Służy do definiowania własnych tagów lub użycie gotowych wykonujących te same akcje wielokrotnie.

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

Zadanie 2

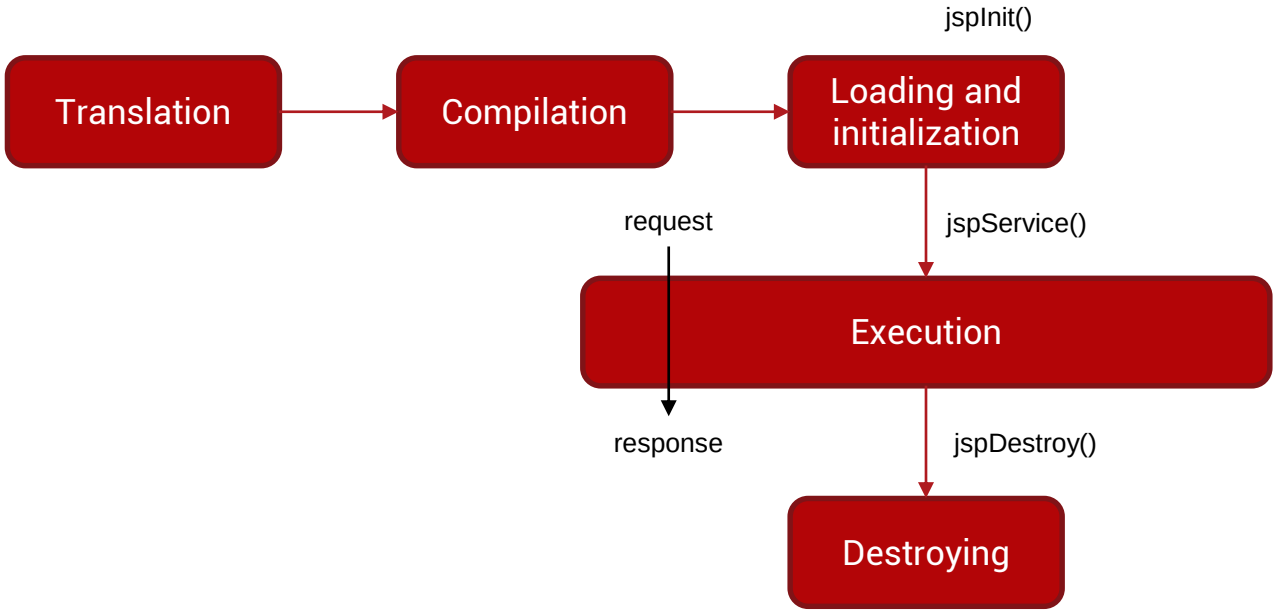
- Utwórz nową stronę menu.jsp
- Przenieś wcześniej stworzony kod odpowiedzialny za wyświetlenie menu do nowego pliku.
- Z wykorzystaniem dyrektywy include połącz strony index.jsp z menu.jsp

Jak wygląda struktura HTML Twojego kodu?

3.
JSP: Scripting Tags

Kod Java na stronie HTML

JSP
lifecycle - powtórka



Java
JSP

Istnieją trzy możliwości osadzenia kodu Java w kodzie strony JSP:

- scriptlet tag <% … %>
- declarative tag <%! … %>
- expression tag <%= … %>

JSP
scriptlet

Realizowany w ramach implementacji **jspService**, a zatem posiada lokalny zakres!
Przy każdym nowym żądaniu będzie wykonywany ponownie.

JSP
declarative

Realizowany w ramach implementacji **jspInit** oraz **jspDestroy**, a zatem posiada zakres całej klasy.

JSP
expression tag

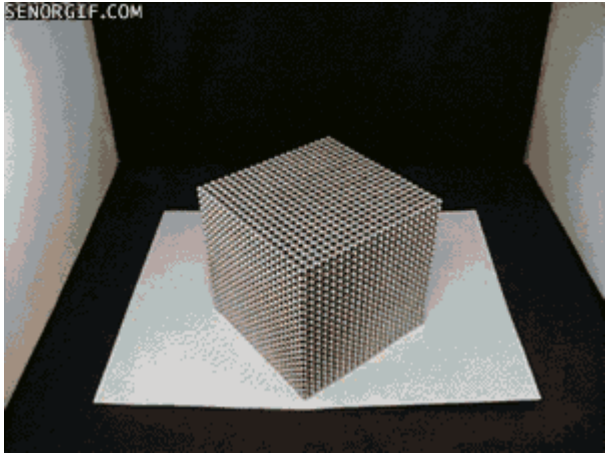
Wykonuje wyrażenie języka Java. Rezultat wyrażenia wkleja jako tekst do kodu HTML strony JSP.

Zadanie 3

- Na stronie index.jsp wyświetl datę i godzinę wyrenderowania użytkownikowi strony
- Przeprowadź eksperyment: pobierz aktualną datę w tagu scriptlet oraz porównaj zachowanie przy pobieraniu daty w tagu declarative.
- Użyj do tego LocalDateTime / DateTimeFormatter

Scripting tags
UNIKAJ

Unikaj umieszczania kodu Java na stronie JSP.
Zapoznaj się z dokumentacją JSTL oraz EL w celu realizacji wymaganych operacji na stronie JSP.



4.
JSP: POST

Obsługa formularzy

Servlet: obsługa POST

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/first-servlet")
public class FirstServlet extends HttpServlet {

 @Override
 protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

 }
}
```

JSP
Obsługa POST

Obsługa komunikacji POST z korzystaniem JSP/Servlets jest bardzo prosta!

W ramach strony JSP tworzymy w HTMLu standardowy formularz. Jako akcję definiujemy nazwę kontekstu naszego servletu, który obsłuży request POST.

Zadanie 4

- Utwórz nową stronę JSP: add-user.jsp
- Stwórz formularz dodawania użytkownika
- Nie zapomnij o określeniu metody komunikacji na POST
- Skieruj akcję formularza na servlet z pracy domowej: AddUserServlet
- Wprowadź wymagane modyfikacje do servletu AddUserServlet tak aby obsługiwał jednocześnie komunikację GET jak i POST.

5.
EL & JSTL

Kodowanie JSP

Obsługa JSP
JSTL

JSTL (JSP Standard Tag Library)
Standardowa biblioteka oferująca tagi do kontroli operacji na stronie JSP.

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

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

EL & JSTL
Dlaczego używać?

Zapewnione:

- escape'owane specjalne znaki XML
- Możliwość wyświetlenia wartości domyślnej jeśli ta przekazana jest nullem.

Obsługa JSP
EL

Praktycznie zawsze w połączeniu z JSTL.
Umożliwia używanie wyrażeń w kodzie JSP.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
```

Podstawy
EL

Wyrażenia w EL realizowane są za pomocą znaków **$** oraz **{}**

Operatory arytmetyczne: **+, -, /, *, %**

Operatory relacji: **>, <, >=, <=, ==, !=**

Operatory logiczne: **&&, ||, !**

EL
przykłady

${sessionScoped.name}

${name}

${2*5}

Flow aplikacji
servlet

W metodach doGet/doPost/… przygotowujemy request lub sesję:

request – jest dostępny w metodzie
Sesję otrzymujemy:

```
req.getSession()
```

Ustawiamy atrybuty do przekazania:

```
req.setAttribute("name", req.getParameter("name"));
req.getSession().setAttribute("sessionName", req.getParameter("name"));
```

Flow aplikacji
servlet - dispatcher

```
RequestDispatcher requestDispatcher = req.getRequestDispatcher("/welcome-user.jsp");
requestDispatcher.forward(req, resp);
```

Flow aplikacji
JSP

```
<body>
Hello ${name}!
Hello ${sessionScope.sessionName}!
</body>
```

Zadanie 5

- Przygotuj nową stronę JSP welcome-user.jsp
- Zmodyfikuj działanie servletu WelcomeUserServlet tak aby przesyłał do JSP imię wykorzystując zakres requestu oraz sesji.
- Wyświetl dwa parametry w HTML na stronie JSP
- Dodaj nowy parametr GET salary który będzie przekazywany do JSP. Wyświetl wartość parametru na stronie.

JSTL
zmienne w JSP

Istnieje możliwość bez użycia Javy definiować wartości nowych zmiennych w JSP:

```
<c:set var="salary" scope="request" value="${1000}"/>
```

JSTL
wypisywanie w JSP

Zamiast bezpośredniego wypisywania tekstu, możemy użyć <c:out>

```
<c:out value="some value" default="some default value" />
```

Zadanie 6

- Nadal pracujemy z plikiem welcome-user.jsp
- Zdefiniuj nową zmienną w JSP o nazwie bonus, która będzie wyliczać 20% bonusu dla wartości przekazanej w parametrze salary. Do zmiennej bonus przypisz wartość salary+wyliczony bonus.

JSTL
sterowanie

- Pętla po kolekcji danych:

```
<c:forEach var="entry" items="elements">
    Element: <c:out value="${entry}"/>
</c:forEach>
```

Dla kolekcji typy key-value entry będzie posiadało dwa pola: key oraz value.

Zadanie 7

- Przygotuj nowy widok users-list.jsp
- W zadaniu wykorzystaj StatisticsServlet z zadania domowego
- W nowym widoku wyświetl listę użytkowników: ich dane + liczbę wyświetleń

JSTL
sterowanie

- Sterowanie warunkami:

```
<c:if test = "${salary > 2000}">
```

```
<c:if test = "${not empty error}">
```

```
<c:choose>
    <c:when test="${isLogged == false}">
        ...
    </c:when>
    <c:otherwise>
        ...
    </c:otherwise>
</c:choose>
```

Zadanie 8

- Na liście użytkowników jeśli wartość gender jest Gender.MAN, wyświetl cały rekord użytkownika w kolorze niebieskim.

JSTL
sterowanie

```
<c:out value="${message}"/>

<c:remove var="salary"/>

<c:forEach var="i" begin="1" end="5"></c:forEach>

<c:forTokens items="Jan,Adam,Anna" delims="," var="name"></c:forTokens>

<c:url value="/index.jsp" var="myURL">
    <c:param name="ID" value="1234"/>
    <c:param name="type" value="list"/>
</c:url>
```

Zadanie 9

- Przygotuj nowy widok login.jsp z jednoelementowym formularzem (pole username).
- Przygotuj nowy servlet LoginServlet, który przyjmując dane POST z formularza JSP ustawi parametr sesji [„username"=>"wartość_pola_formularza"]
- Dodaj link do widoku login.jsp w menu

6.
Filters

Filtrujemy komunikację

Filter
założenia

Filtry wykorzystywane w servletach oraz JSP są klasami, używanymi do filtrowania i podejmowania akcji w komunikacji client-backend oraz modyfikowania odpowiedzi backendu zanim zostanie wysłana do klienta.

Filter
przeznaczenie

Przykładowe/sugerowane przeznaczenie filtrów:

- Autentykacja
- Autoryzacja
- Szyfrowanie
- Kompresja danych

Filter
syntax

```
@WebFilter(
        filterName = "AuthenticationFilter",
        urlPatterns = {"/*"},
        initParams = {
                @WebInitParam(name = "allowedUser", value = "root")
        })
public class AuthenticationFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
throws IOException, ServletException {

    }

    @Override
    public void destroy() {

    }
}
```

Zadanie 10

- Napisz filtr AuthenticationFilter który sprawdzi czy username sesji jest równy nazwie użytkownika z parametrów inicjujących ten filtr. Jeśli tak, ustaw sesyjny atrybut isLogged jako true. W przeciwnym wypadku – false.
- Wyświetlaj na górze strony index.jsp: jeśli jesteśmy zalogowani „Zalogowany jako: [username]". Jeśli niezalogowany, wyświetlaj napis „[zaloguj]", również prowadzący do widoku login.jsp. Wykorzystaj do tego atrybut isLogged

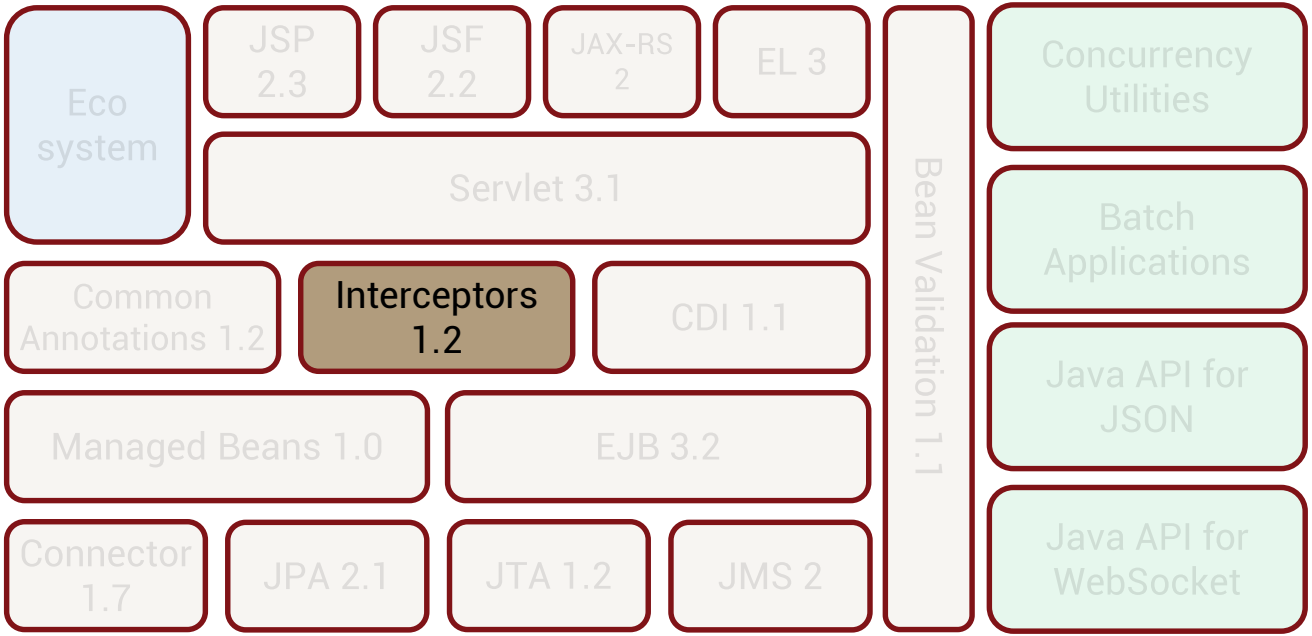
Szybka powtórka: Zadanie 11

- Dla servletu AddUserServlet przygotuj obsługę komunikatów i błędów. Jeśli dodawanie użytkownika się powiodło, wyświetl na stronie index.jsp kolorem zielonym informację o udanej operacji, jeśli pojawił się błąd, na stronie index.jsp wyświetl informację kolorem czerwonym o wystąpieniu błędu – jakiego?

7.
JEE: Interceptors

Oddzielamy niezależne funkcje

JEE 7


Eco system
JSP 2.3
JSF 2.2
JAX-RS 2
EL 3
Servlet 3.1
Common Annotations 1.2
Interceptors 1.2
CDI 1.1
Managed Beans 1.0
EJB 3.2
Connector 1.7
JPA 2.1
JTA 1.2
JMS 2
Bean Validation 1.1
Concurrency Utilities
Batch Applications
Java API for JSON
Java API for WebSocket

Interceptor
AOP

Aspect Oriented Programming – sposób tworzenia aplikacji polegający na jak najbardziej szczegółowym separowaniu elementów niezależnym względem siebie.

Interceptory w Javie realizują podejście AOP.

Interceptor
obserwator

Interceptor to swoisty obserwator. Przejmuje sterowanie zadaniem jak tylko zostanie wywołany w przypadku wywołania obserwowanego bytu (np. metody).

Interceptor
syntax

```
public class AddUserInterceptor {

    Logger logger = Logger.getLogger(AddUserInterceptor.class.getName());

    @AroundInvoke
    public Object intercept(InvocationContext context) throws Exception {
        logger.info("Add user has been invoked!");
        return context.proceed();
    }
}
```

```
@Override
@Interceptors(AddUserInterceptor.class)
public void addUser(User user) {
    UsersRepository.getRepository().add(user);
}
```

Zadanie 12

- Napisz interceptor, który przy braku ustawionej płci użytkownika będzie zgadywał i ustawiał ją automatycznie.
- Załóż obsługę tylko polskich imion, gdzie imiona kończące się na literę „a" to imiona damskie, pozostałe to imiona męskie.
- Dodatkowo wykonywaną akcję będzie logował na poziomie INFO do logów aplikacji.
- Pamiętaj aby select w formularzu dodawania użytkowników miał pierwszą opcję pustą, która niewybrana ustawia wartość gender=null w obiekcie User

8.
Usprawniamy development

Szybciej, sprawniej, wydajniej

Wildfly: IntelliJ & Maven
wildfly-maven-plugin

Istnieje możliwość szybkiego zarządzania artefaktami na serwerze wykorzystując WildFly'a (10.1.0) dostarczonego w postaci pluginu maven:

```
<plugin>
    <groupId>org.wildfly.plugins</groupId>
    <artifactId>wildfly-maven-plugin</artifactId>
    <version>1.2.1.Final</version>
</plugin>
```

wildfly-maven-plugin
debugger

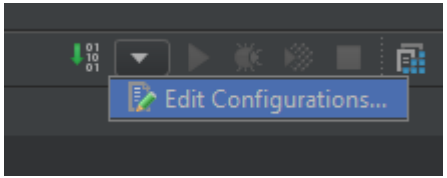
Istnieje również możliwość dodania konfiguracji, która będzie uruchamiała server z możliwością debuggera:

```
<plugin>
    <groupId>org.wildfly.plugins</groupId>
    <artifactId>wildfly-maven-plugin</artifactId>
    <version>1.2.1.Final</version>
    <configuration>
        <java-opts>
            <java-opt>-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005</java-opt>
        </java-opts>
    </configuration>
</plugin>
```

Zadanie 13: Nowa konfiguracja

Uwaga! Zanim rozpoczniesz upewnij się, że Twoja instancja WildFly'a została całkowicie zamknięta.

Za pomocą IntelliJ, przejdź do edycji konfiguracji:



Następnie ikoną + dodaj nową konfigurację Jboss / Local
Jeśli Jboss nie widnieje na liście, wybierz „items more" mieszczące się w dole listy.

Konfiguracja
Jboss

W pozycji **Application Server** wybieramy **Configure**.
Jako **Jboss Home** wskazujemy główny folder **wildfly**.

W pozycji **After launch** wybieramy docelową przeglądarkę gdzie będą ładowane nowe wersje artefaktów.

W zakładce **Deployment**, dodajemy nowy artefakt helloexample.war

Jeśli nie pojawiła się nam nowa konfiguracja oznacza to, że była to konfiguracja domyślnych wartości serwera, ponownie wykonujemy operację dodania nowej konfiguracji z wykorzystaniem Jboss/Local.

Dodatkowo możemy nadać nazwę naszej konfiguracji.

Thanks!!

Q?