Hello!

Rafał Misiak

Java Developer dla Stibo Systems (DK) w Ciklum

slack: @rafalmisiak

rafalmisiak@gmail.com

JEE
Podstawy

Harmonogram
Java EE

- Servlets
- EJB

- JSP (w tym również rozszerzenie servletów)
- Dispatcher
- ManagedBeans
- Interceptors
- Odrobinę o JEE8
- Więcej o poznanych komponentach

1.
JEE – Java Enterprise Edition

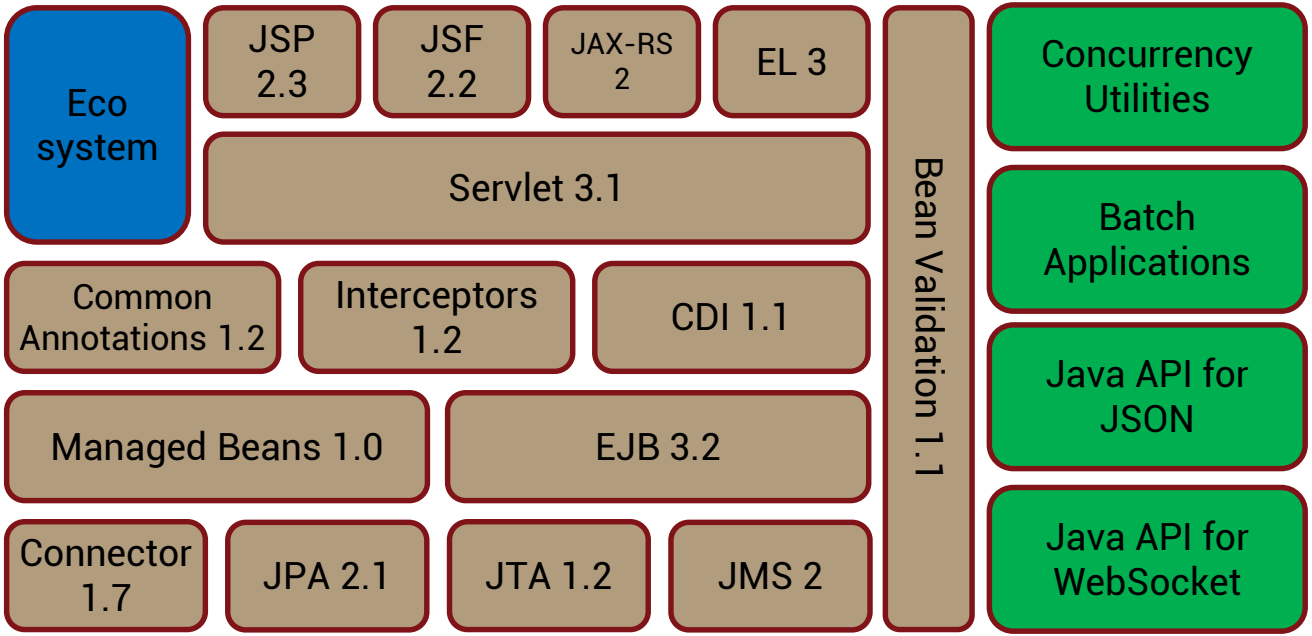
„Korporacyjne" rozszerzenie Java SE

Java Enterprise Edition
Charakterystyka

- Serwerowa platforma programistyczna
- Definiuje standard oparty na wielowarstwowej architekturze komponentowej
- Określa zbiór interfejsów jakich implementację musi dostarczać zgodny serwer aplikacyjny
- Specyfikacja zestawu API dla Javy ma na celu usprawnić wytwarzanie komercyjnego oprogramowania

Java Enterprise Edition
Ewolucja

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

Java Enterprise Edition
Model

- **JSP** – proste strony łączące technologie frontendowe z Javą, kompilowane do servletów
- **JSF** – framework MVC, przykrywa servlety, udostępnia między innymi szablonowanie
- **JAX-RS** – api restowego webserwisu
- **EL** (expression language) – używany w warstwie webowej, specjalny język programowania używany między innymi w JSP umożliwiający dynamikę stron
- **Servlet** – serwerowa klasa obsługująca komunikację żądanie-odpowiedź

Java Enterprise Edition
Model

- **Common Annotations** –zbiór wspólnych adnotacji dla JSE oraz JEE
- **Interceptors** – wzorzec polegający na zmianie standardowego flow aplikacji
- **CDI** – mechanizm wstrzykiwania zasobów
- **Managed Beans** – specjalne klasy pozwalające na dostęp oraz manipulację danymi
- **EJB** – serwerowe klasy pozwalające na implementację logiki biznesowej

Java Enterprise Edition
Model

- **JPA** –API definiujące połączenia do baz danych
- **JTA** – API definiujące transakcyjność aplikacji
- **JMS** – API definiujące mechanizm sterowany komunikatami
- **Bean Validation** – zbiór adnotacji pozwalających na natychmiastową weryfikacji poprawności danych, np. @NotNull, @Size(min,max)

Java Standard Edition vs Java Enterprise Edition

Java to zarówno język programowania jak i platforma w jednym.

Java jest wysoko poziomowym, zorientowanym obiektowo językiem programowania.

Platforma Java jest środowiskiem uruchomieniowym dla aplikacji napisanych w języku Java.

Java Standard Edition vs Java Enterprise Edition

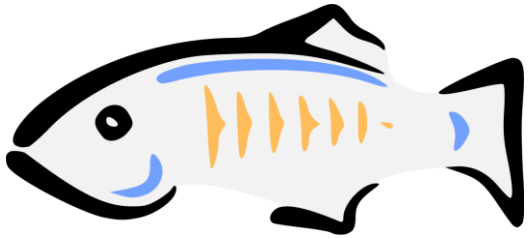
JSE dostarcza podstawowe funkcjonalności, definiuje wszystko od podstawowych typów, obiektów po rozbudowane klasy, które są używane komunikacji sieciowej, tworzenia zabezpieczeń, dostępu do baz danych, parsowania XML, tworzenia GUI, itp.

JEE jest rozszerzeniem dla JSE. Dostarcza **API** oraz środowisko uruchomieniowe dla aplikacji budowanych na wielką skalę, wielowarstwowych, skalowalnych, *niezawodnych*.

2.
Aplikacje JEE na start!

Serwery JEE

Szeroki wachlarz
serwerów JEE



ORACLE
WEBLOGIC

WildFly

jetty://

IBM
WebSphere

Serwer JEE

Serwer na którym aplikacja jest **deployowana** zapewnia pełne zarządzanie cyklem życia aplikacji, jej skalowalnością, **dostarcza implementację** JEE API.

Nasz wybór:

Zadanie: Narzędzia pomocnicze
Java

```
# java -version
# echo $JAVA_HOME
```

Pusto?

```
# cd
# nano .bash_profile
# export JAVA_HOME=/usr/lib/jvm/default-
java
# source ~/.bash_profile
```

Zadanie: Instalacja WildFly

```
$ cd
$ wget
http://download.jboss.org/wildfly/11.0.0.Final/wildfly-11.0.0.Final.tar.gz
$ tar -zxvf wildfly-11.0.0.Final.tar.gz
$ ln -s wildfly-11.0.0.Final wildfly

$ nano ~/.bash_profile
$ export JBOSS_HOME=~/wildfly
$ export WILDFLY_HOME=$JBOSS_HOME

# source ~/.bash_profile
```

Do pliku .bashrc należy dodać:

```
. ~/.bash_profile
```

Zadanie: Nowy użytkownik Wildfly

```
# ./bin/add-user.sh
```

Dokonujemy wyboru (a) Management User

Nadajemy własną nazwę i hasło.

Zezwalamy na dostęp do zdalnego API.

I gotowe!

Odwiedźmy: 127.0.0.1:9990

Zadanie: Uruchom serwer

```
$ cd $WILDFLY_HOME
./bin/standalone.sh
```

Adresy:
127.0.0.1:8080
127.0.0.1:9990

Powinny odpowiedzieć odpowiednio: domyślną stroną startową Wildfly oraz konsolką administracyjną z monitem o zalogowanie się.

3.
Aplikacja JEE

Maven support

Plugin Maven
maven-[war | jar]-plugin

Pluginy umożliwiają kompilację zbudowanie wara lub odpowiednio jara.

To, do jakiego pliku ostatecznie nasza aplikacja zostanie zapakowana, określa konfiguracja:

<packaging>jar</packaging>

lub

<packaging>war</packaging>

Podstawowe różnice
war vs jar (vs ear)

- **.jar (Java Archive)** – zawiera biblioteki, dodatkowe zasoby, pliki konfiguracyjne, backendową logikę aplikacji

- **.war (Web Application Archive)** – zawiera warstwę webową aplikacji, może ona zostać zdeployowana w konterze servletowym/jsp. Zawiera najczęściej kod jsp, html, javascript jak również dodatkowe kontrolery zarządzające tą warstwą aplikacji napisane już w języku Java.

- **.ear (Enterprise Application Archive)** – zawiera jeden lub więcej moduły, używany do deploymentu bardziej złożonych aplikacji w postaci jednej paczki, która zawiera wszystkie swoje składowe

Plugin Maven
maven-[war | jar]-plugin

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.0</version>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
    </plugin>
</plugins>
```

Zadanie 2.1

- Utwórz projekt Maven z wykorzystaniem archetypu: maven-archetype-webapp o nazwie "search-engine„
- Utwórz katalog "java" w drzewie projektu "main„
- Ustaw katalog "java" jako "Sources Root„
- Stwórz pakiet com.infoshareacademy.searchengine
- Wykorzystaj plugin maven-jar-plugin oraz packaging jar
- Stwórz klasę "Main" z metodą "main" wyświetlającą do konsoli "Hello World!". Zbuduj, uruchom projekt w konsoli.

Zadanie 2.2

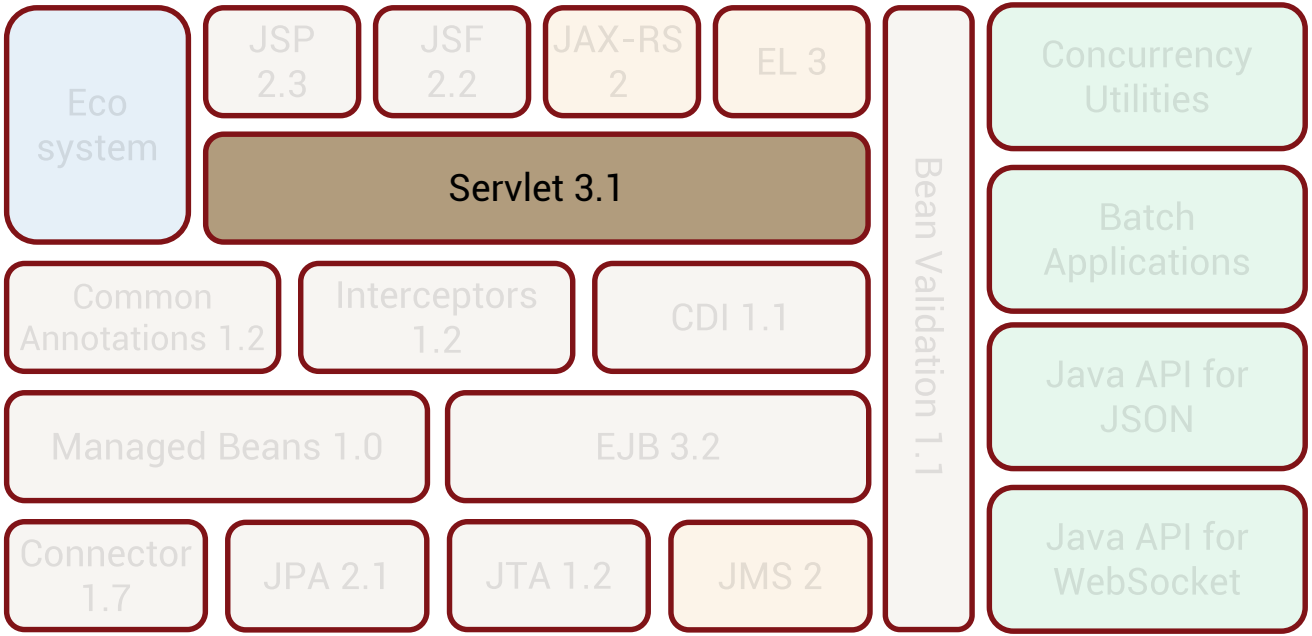
- Utwórz pod-pakiet „ searchengine.domain"
- Stwórz w nim klasę User z polami: id, name, surname, login, age. Zapewnij gettery i settery dla wskazanych pól
- Utwórz pod-pakiet „searchengine.dao"
- Stwórz w nim interfejs UsersRepositoryDao z metodami: addUser, getUserById, getUserByLogin, getUsersList
- W podpakiecie „dao" utwórz klasę UsersRepositoryDaoBean implementującą interfejs UsersRepositoryDao. Zaimplementuj wymagane metody wykorzystując klasę com.infoshareacademy.searchengine.repository.UsersRepository

- Wykorzystaj klasę Main do wyświetlenia imion wszystkich użytkowników repozytorium. Użyj DAO! Repozytorium umieść w podpakiecie searchengine.repository
- Dlaczego używamy klasy UsersRepository ?

4.
Servlet

Request-Response Communication

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

Specyfikacja
Java Servlet API

Specyfikuje klasy odpowiedzialne za obsługę requestów HTTP

Servlet API to dwa kluczowe pakiety:

javax.servlet – zawiera klasy i interfejsy stanowiące kontrakt pomiędzy klasą servletu, a środowiskiem uruchomieniowym

javax.servlet.http – zawierający klasy i interfejsy stanowiące kontrakt między klasą servletu, a środowiskiem uruchomieniowym gdzie komunikacja odbywa się w protokole HTTP

Zarządzanie
Kontener Webowy

Zarządzaniem Servletami zajmuje się część serwera aplikacji zwanego **kontenerem** webowym.

Z oparciu o nasz wybór, w Wildfly kontenerem webowym jest Undertow, którego konfigurację możemy znaleźć na liście subsystemów serwera.

Zależności do pracy z Servletem

```
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>8.0</version>
    </dependency>
<dependencies>
```

Rozpoczęcie pracy z Servletem

```
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 doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        // provide your code here
    }
}
```

Zwrócenie tekstu możliwe jest za pomocą PrintWirter'a, którym piszemy do odpowiedzi:

```
PrintWriter writer = resp.getWriter();
writer.println("Hello World!");
```

Zadanie 3

- Stwórz podpakiet searchengine.servlets – tutaj umieszczaj wszystkie kolejne servlety
- Stwórz pierwszy servlet o nazwie HelloServlet w kontekście „hello-servlet"
- Spraw aby wyświetlał on nazwę Hello World!
- Wykorzystaj plugin maven-war-plugin oraz packaging war
- Zbuduj projekt, utwórz paczkę war dla projektu
- Wykonaj deploy aplikacji na serwerze
- Uruchom w przeglądarce

Zarządzanie
App root

Pierwszą opcją jest zmiana domyślnej wartości app root czyli podstawowej ścieżki dostępu do naszej aplikacji przez przeglądarkę.

Nazwą tą zarządzamy w pliku pom.xml:

```
<build>
    …
    <finalName>${project.artifactId}</finalName>
    …
</build>
```

Zarządzanie
Context root

Kolejną opcją jest możliwość ustawienia domyślnego context root jako naszej aplikacji, czyli zamiast odwołania:

http://server-name:port/app-root/index.jsp

Odwołamy się:

http://server-name:port/index.jsp

W tym celu definiujemy plik jboss-web.xml

jboss-web.xml
Context root

Plik jboss-web.xml tworzymy w katalogu WEB-INF:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="
      http://www.jboss.com/xml/ns/javaee
      http://www.jboss.org/j2ee/schema/jboss-web_11_0.xsd" version="11.0">
    <context-root>/</context-root>
</jboss-web>
```

Zadanie 4

Skonfiguruj aplikację tak aby uruchamiana była z domyślnego kontekstu / bez konieczności dodawania nazwy aplikacji w ścieżce

Servlet
Obsługa requestu

Servlet wrażliwy na zadane parametry requestu.

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    // provide your code here
}
```

Zmienna lokalna *req* zawiera sporo pomocnych nam informacji.
Między innymi możemy ją wykorzystać do pobrania danych requestu.

```
req.getParameter("name")
```

Pobierze nam wartość parametru „name" z adresu w przeglądarce.

Servlet
Obsługa odpowiedzi

Za pomocą Servletu również możemy generować odpowiedzi.

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    // provide your code here
}
```

Zmienna lokalna *resp* pozwala na generowanie odpowiedzi. Za pomocą:

```
resp.setContentType("text/html;charset=UTF-8");
PrintWriter writer = resp.getWriter();

writer.println("<!DOCTYPE html>");
```

Możemy ustawić kodowanie strony jak również pobrać writera, który będziemy pisać kod wynikowy.

Zadanie 5

- Przygotuj servlet WelcomeUserServlet w kontekście „welcome-user" który wyświetli napis "Hello :name!" gdzie :name to wartość parametru z requestu.
- Opakuj to zdanie w prostego HTML'a: <!DOCTYPE html><html><body>…</body></html>
- Jeśli parametr name nie został podany w requescie, zwróć status BAD_REQUEST – wykorzystaj do tego klasę ze statycznymi kodami HttpServletResponse

Zadanie 6

- Utwórz nowy servlet FindUserByIdServlet w kontekście „find-user-by-id"
- Wykonaj wyszukiwanie użytkownika po zadanym w requeście parametrze „id".
- Uwzględnij warunki z zadania 5.
- Do rozwiązania wykorzystaj klasy DAO, domain, repository

Zadanie 7: Servlet
Statystyki

Configuration -> Subsystems -> Web/HTTP – Undertow -> HTTP -> View -> Edit -> Statistics enabled=true

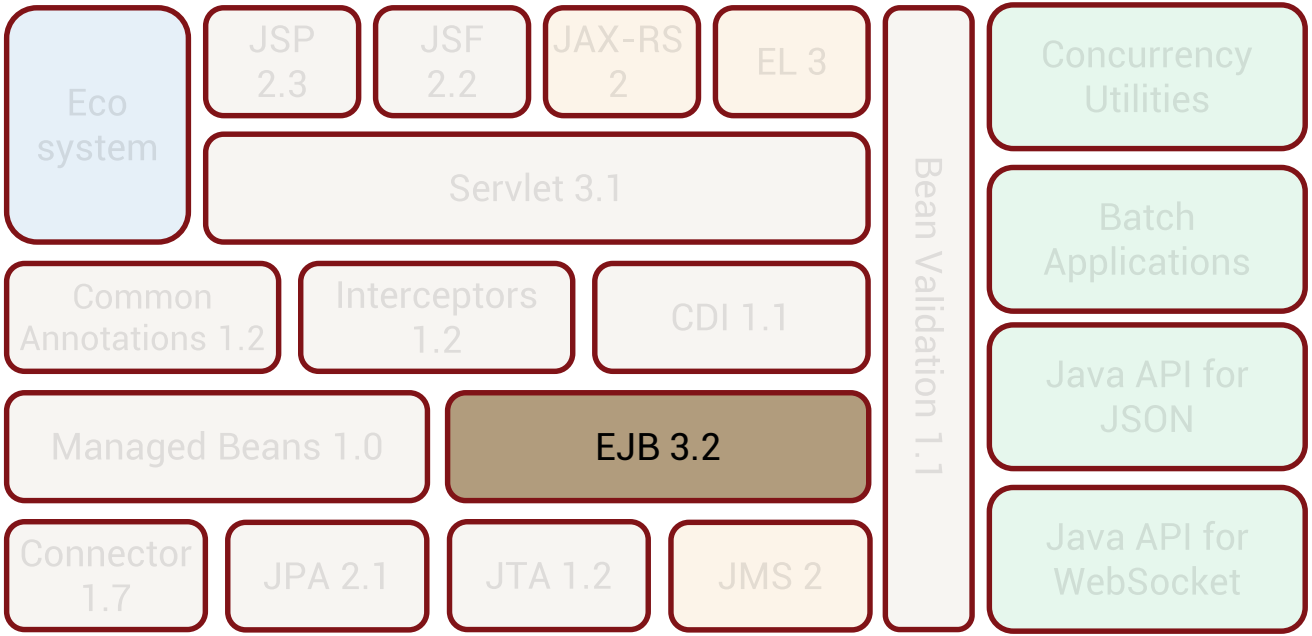
Podgląd możliwy w:

Deployments -> [:artefakt] -> View -> subsystem -> undertow -> servlet -> [:servlet]

5.
EJB

Enterprise Java Bean

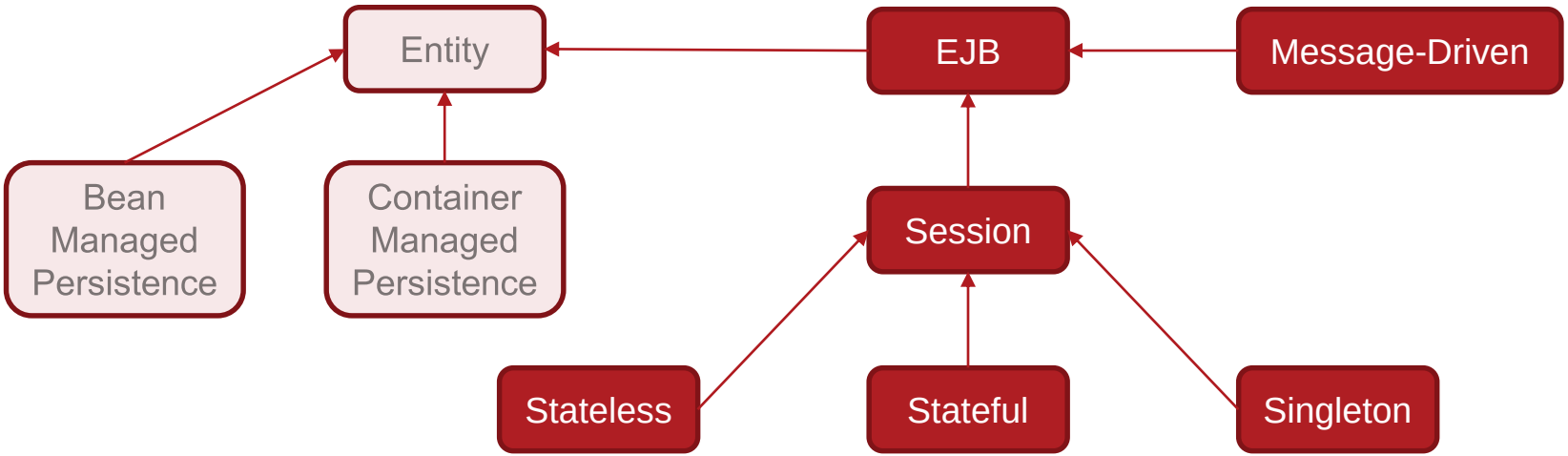
JEE 7


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

Czym jest EJB ?

- Jedna z najpopularniejszych części JEE
- Opisuje logikę biznesową aplikacji
- Zarządzana przez kontener EJB
- Udostępnia usługi:
  - transakcyjność
  - trwałość
  - rozproszenie
  - odseparowanie warstwy prezentacji od logiki biznesowej
  - skalowalność

Ogólna budowa
EJB



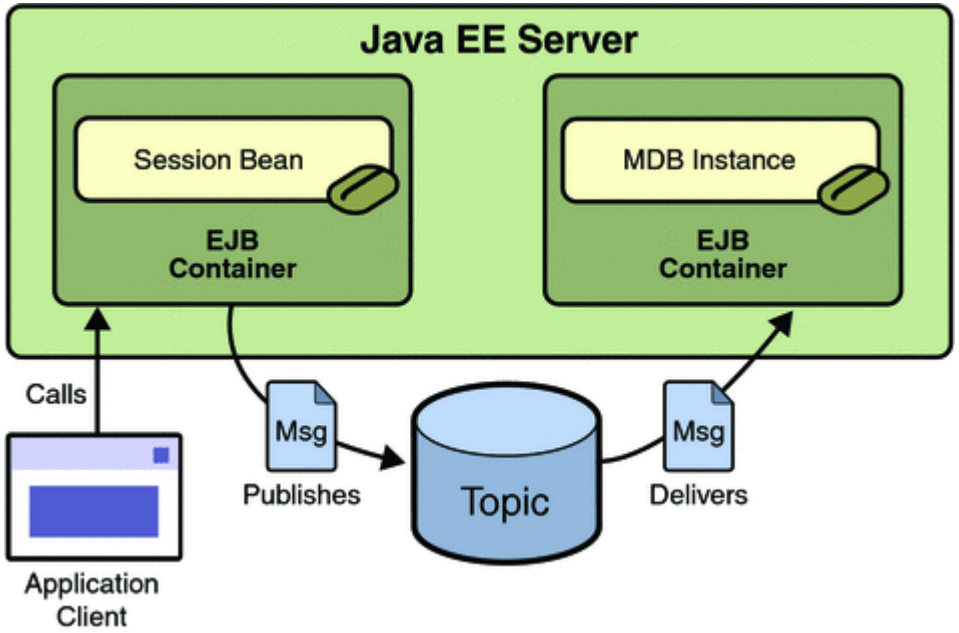
EJB
Message-Driven

Komponent sterowany wiadomościami, często opierający się na mechanizmie JMS (część specyfikacji JEE).

Komponent ten, nie jest wywoływany bezpośrednio przez klienta. Reaguje na wiadomości umieszczone, np. w kolejce.

Taka obsługa pozwala na podejście całkowicie asynchroniczne.

EJB
Message-Driven



źródło: oracle.com

EJB
Session

Sesyjny komponent EJB realizuje konkretne zadania klienta. Klient zleca komponentowi wykonanie jakiegoś zadania poprzez wywołanie metody dostępnej w interfejsie tego komponentu.

Sesyjny komponent może obsługiwać tylko jednego klienta na raz.

Stan sesyjnego komponentu nie wykracza poza sesję, jego stan nie jest utrwalany, np. w bazie danych.

Session EJB
Stanowość

Stateful (stanowy) – pamięta stan dla konkretnej sesji z klientem, „stan konwersacji" z klientem, obejmujący wiele wywołań metod

Stateless (bezstanowy) – nie pamięta konwersacji z klientem, nie zachowuje swojego stanu nawet na czas trwania jednej sesji, jego stan jest zachowany na czas wywołania jednej metody, serwer nie daje gwarancji utrzymania tego stanu na dalszym etapie komunikacji

Singleton – istnieje tylko jeden stan (jedna instancja) w skali całej aplikacji, bez względu na to ilu klientów zostanie do niego podłączonych

Session EJB
Stanowość

```
import javax.ejb.Stateless;
import java.util.List;

@Stateless
public class UsersRepositoryDaoBean implements UsersRepositoryDao {

    …

}
```

Zasięg EJB
@Local / @Remote

Adnotacja @Local oznacza, że metody będzie można wywoływać maksymalnie z innego modułu jednak mieszczące się w tej samej paczce EAR

Adnotacja @Remote oznacza, że metody będzie można wywoływać nie tylko tak jak w @Local ale również z całkowicie niezależnego modułu mieszczącego się zarówno na tym samym serwerze jak również na zdalnej maszynie.

Zasięg EJB
@Local / @Remote

```
import javax.ejb.Local;

@Local
public interface UsersRepositoryDao {

    …

}
```

Zadanie 8

- Przekonwertuj interfejs oraz klasę DAO na bezstanowe EJB o zakresie lokalnym.

EJB
Dependency Injection

Do tej pory poprawnie zdefiniowaliśmy nasze beany. Ale jak ich użyć w naszej aplikacji?

Dependency Injection jako wzorzec architektury oprogramowania.

Charakteryzuje się architekturą pluginów zamiast jawnego tworzenia bezpośrednich zależności między klasami.

Polega na przekazywaniu między obiektami gotowych, ustanowionych obiektów danych klas (beanów).

EJB
Dependency Injection

Bez użycia kontenera aplikacji JEE możemy DI zrealizować za pomocą konstruktora.

```
public class User {
    private final static Permissions permissions;

    public User() {
        this.permissions = new Permissions();
    }
}
```

```
public class User {
    private final static Permissions permissions;

    public User(Permissions permissions) {
        this.permissions = permissions;
    }
}
```

EJB
Dependency Injection

Istnieje możliwość przekazania zarządzania zależnościami naszemu kontenerowi aplikacji JEE. Takie działanie nazywane jest Inversion of Control i jest jednym ze wzorców projektowych.

Takie podejście zapewnia nam **loose coupling**, które ma na celu jak najmniejsze powiązanie obiektów między sobą.

Kontener aplikacji JEE dostarcza nam mechanizm nazywany wstrzykiwaniem zależności.

Kontener posiadający funkcjonalność wstrzykiwania nazywany jest kontenerem DI lub IoC. EJB jest kontenerem DI.

EJB
Dependency Injection

Używając kontenera aplikacji JEE natomiast, możemy użyć adnotacji @EJB.

```
public class User {

    @EJB
    PermissionsInterface permissions;

    public User() {
    }
}
```

Zadanie 9

- Przekształć servlet FindUserByIdServlet na postać korzystającą z EJB DI.

CDI
Context and Dependency Injection

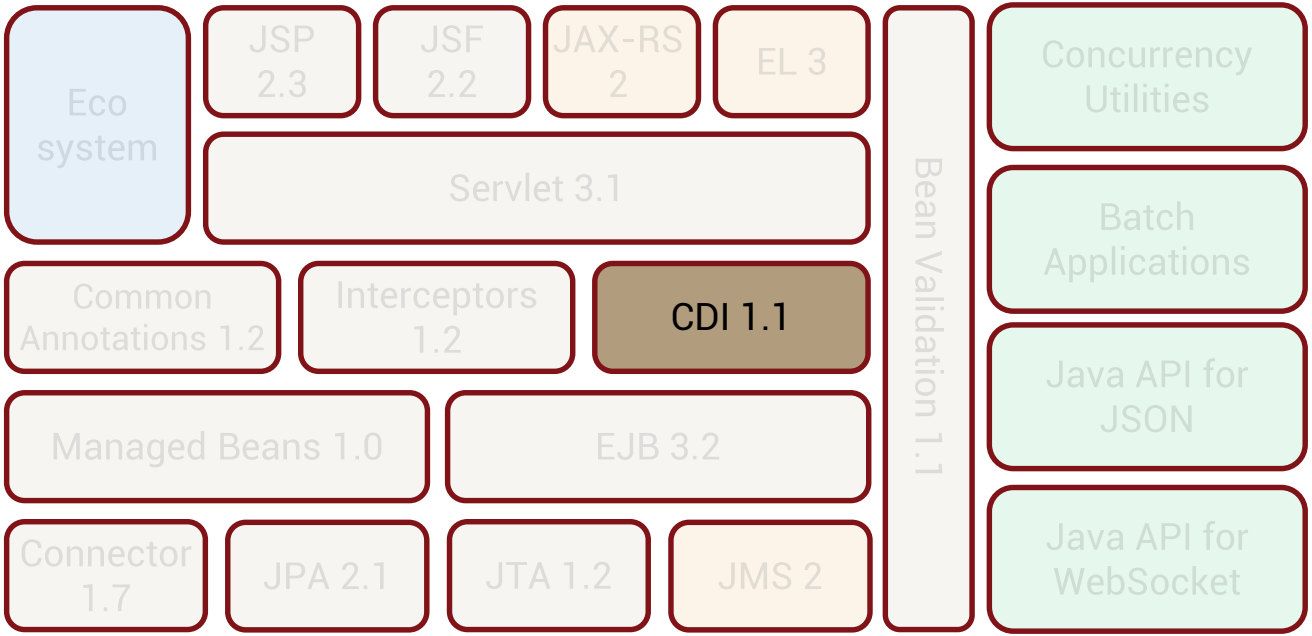
Zestaw usług pozwalający na zachowanie loose coupling między warstwami aplikacji.

Oznaczenia beanów CDI:

@ApplicationScoped, @SessionScoped, @ConversationScoped,
@RequestScoped,
@Interceptor, @Decorator,
@Stereotype,
@Dependent

Pozwalana wstrzyknięcie większości obiektów występujących w ramach aplikacji. Nie muszą być one EJB.

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

CDI
Context and Dependency Injection

Przewagą EJB nad CDI będzie fakt, że kontener przejmie kontrolę nad transakcjami, bezpieczeństwem, współbieżnością, pulami obiektów.

EJB
@Inject / @EJB

@EJB pozwala na wstrzykiwanie tylko i wyłącznie obiektów zarządzanych przez kontener EJB.

@Inject obsługiwana jest przez kontener CDI i pozwala na wstrzykiwanie zarówno obiektów zarządzanych przez kontener EJB jak i pozostałych beanów.

Nawiązując do rady Adama Bienia:
You can use both annotations to inject EJBs. Start with @Inject and if you encounter any problems, switch to @EJB.

EJB
@Inject / @EJB

```java
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Inject {
}
```

```java
@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface EJB {
    String name() default "";

    String description() default "";

    String beanName() default "";

    Class beanInterface() default Object.class;

    String mappedName() default "";

    String lookup() default "";
}
```

Zadanie 10

- Wykonaj eksperyment zamieniając wstrzyknięcie zależności EJB na CDI.

Zadanie 11

- Zaimplementuj pozostałe metody klasy DAO oraz doimplementuj brakujące servlety dla tych metod.
- Wykonaj deploy aplikacji, przetestuj jej działanie.

6.
@Remote

Remote Interface Call

@Remote
idea

- Idea interfejsów zdalnych to możliwość wykonywania metod na naszym EJB spoza naszego modułu.

Zadanie 12

- Stwórz interfejs zdalny dla Repozytorium Użytkowników: UsersRepositoryDaoRemote
- Interfejs powinien zawierać tylko jedną metodę, zwracającą listę imion użytkowników: getUsersNames

Zadanie 13

- Utwórz nowy projekt Maven (nie zamykaj dotychczasowego projektu JEE) o nazwie artifactID:search-engine-standalone
- W celu utworzenia projektu możesz wykorzystać archetyp: org.apache.maven.archetypes:maven-archetype-quickstart
- Zbuduj, uruchom, sprawdź czy aplikacja konsolowa działa poprawnie

@Remote
Biblioteka interfejsu zdalnego

- Aby było możliwe wywołanie zdalne interfejsu należy stworzyć bibliotekę zawierającą ten interfejs. Do tego celu tworzymy bibliotekę z interfejsem korzystając z pluginu: maven-ejb-plugin.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-ejb-plugin</artifactId>
    <version>3.0.0</version>
    <configuration>
        <ejbVersion>3.2</ejbVersion>
        <generateClient>true</generateClient>
        <clientIncludes>
            <clientInclude>com/infoshareacademy/searchengine/dao/UsersRepositoryDaoRemote.class
            </clientInclude>
        </clientIncludes>
    </configuration>
</plugin>
```

Zadanie 14

- Stwórz bibliotekę, która będzie zawierała nowo utworzony interfejs komunikacji zdalnej.
- Zbuduj projekt wykorzystując dotychczasowe znane komendy:
  `mvn clean package`
- Uruchom wtyczkę maven-ejb-plugin
  `mvn ejb:ejb`
- Zainstaluj bibliotekę w lokalnym repozytorium:

```
mvn install:install-file -Dfile=search-engine-client.jar
-DgroupId=com.infoshareacademy -DartifactId=search-
engine-client -Dpackaging=jar -Dversion=1.0
```

Zadanie 15

- Załącz nową bibliotekę search-engine-client jako zależność mavenową w nowym projekcie standalone

```
<dependency>
    <groupId>com.infoshareacademy</groupId>
    <artifactId>search-engine-client</artifactId>
    <version>1.0</version>
</dependency>
```

Zadanie 16

- Zbuduj aplikację w taki sposób aby zawierała wewnątrz wszystkie załączone zależności mavenowe. Do tego celu służy kolejny plugin: maven-assembly-plugin.
- Zbuduj aplikację, zajrzyj do katalogu target
- Sprawdź czy wygenerowany artefakt zawiera zależności:
  `jar –tf <name.jar> | grep "infoshare"`

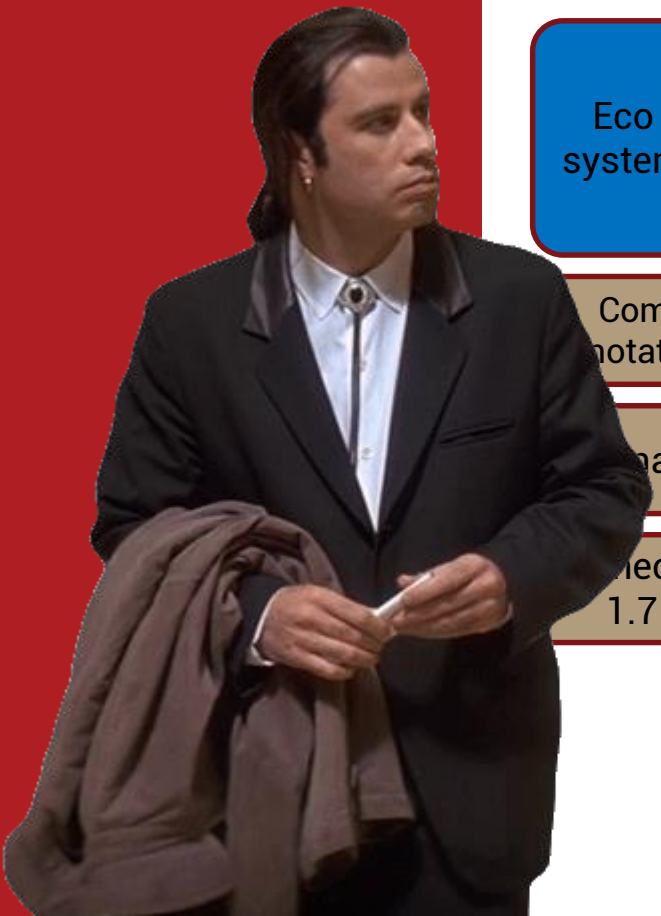
Zadanie 16.1

```
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.1.0</version>
    <configuration>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <classpathPrefix>libs/</classpathPrefix>
                <mainClass>
                    com.infoshareacademy.App
                </mainClass>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

1.
JNDI Lookup

Java Naming and Directory Interface

JEE 7



Eco system

JSP 2.3

JSF 2.2

JAX-RS 2

EL 3

Servlet 3.1

Common notations 1.2

Interceptors 1.2

CDI 1.1

naged Beans 1.0

EJB 3.2

lector 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

JNDI
Charakterystyka

- JNDI nie znajduje się w specyfikacji JEE ale jest mocno wykorzystywane dlatego kontenery aplikacyjne implementują to API
- JNDI służy do wyszukiwania obiektów po ich nazwach
- Wyszukiwanie odbywa się poprzez klasę **InitialContext** i metodę **lookup(String nazwa)**

Zadanie 16

- Przygotuj wymagane zależności w aplikacji standalone. Pamiętaj, że będziemy wykorzystywali implementację dostarczoną przez Wildfly

```
<dependency>
    <groupId>org.wildfly</groupId>
    <artifactId>wildfly-client-all</artifactId>
    <version>11.0.0.Final</version>
</dependency>
```

Zadanie 17

- Przygotuj konfigurację, która będzie spełniała wymagania swojej instancji serwera Wildfly

```
Hashtable<String, String> properties = new Hashtable<String, String>();
properties.put(Context.INITIAL_CONTEXT_FACTORY, "org.jboss.naming.remote.client.InitialContextFactory");
properties.put("jboss.naming.client.ejb.context", "true");
properties.put(Context.PROVIDER_URL, "http-remoting://localhost:8080");
properties.put(Context.SECURITY_PRINCIPAL, "<user>");
properties.put(Context.SECURITY_CREDENTIALS, "<password>");
Context context = new InitialContext(properties);
```

Zadanie 18

- Zaprezentuj w konsoli listę imion użytkowników wykorzystując nowy kontekst.
- JNDI NAME znajdziemy w konsolce WIldFly: Runtime -> Standalone Server -> Subsystems -> JNDI View -> View -> java:jboss/exported

```
UsersRepositoryDaoRemote generator =
        (UsersRepositoryDaoRemote)
                context.lookup(„<JNDI NAME>");
```

Thanks!!

Q?