



# Hello!

## Rafał Misiak

Java Developer dla Stibo Systems (DK) w Ciklum

slack: @rafalmisiak

rafalmisiak@gmail.com

# JEE Podstawy

# Materialy na github

```
git clone  
https://github.com/infoshareacademy/  
jjdd3-materialy-JEE
```

1.

# JEE – Java Enterprise Edition

„Korporacyjne” rozszerzenie Java SE

# Java Enterprise Edition

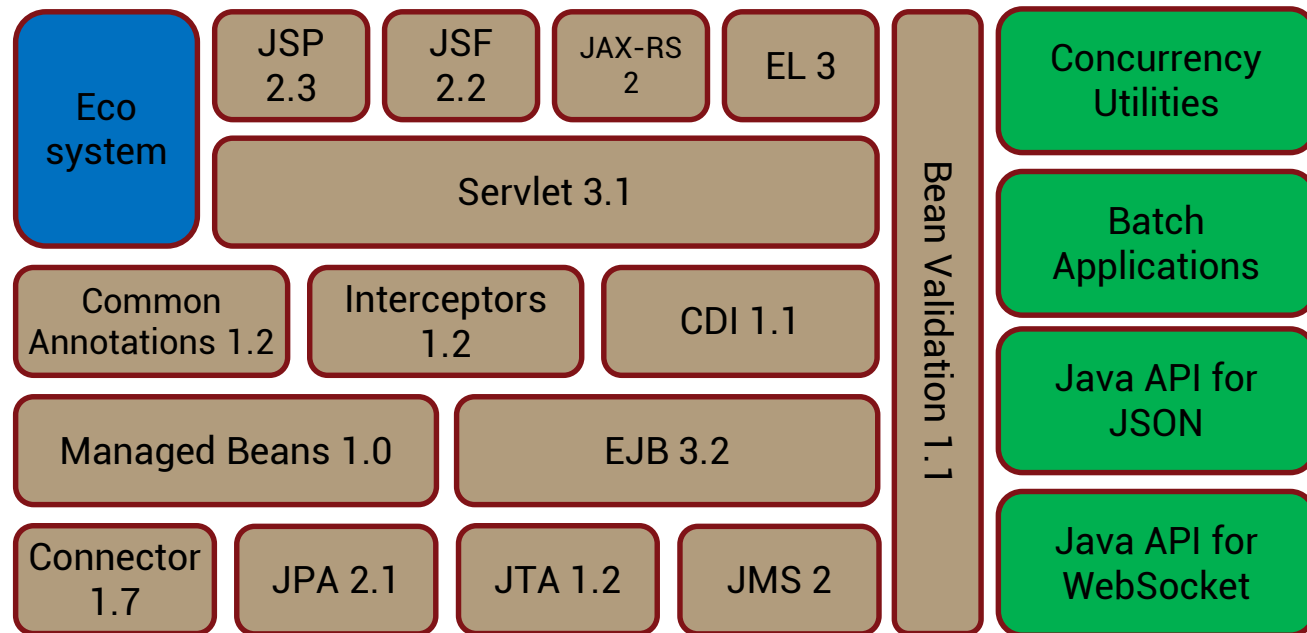
## Charakterystyka

- 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



# Java Enterprise Edition Model

- **JSP** – *JavaServer Pages* – tworzenie dynamicznych dokumentów webowych
- **JSF** – *JavaServer Faces* – framework MVC, upraszcza tworzenie interfejsów użytkownika
- **JAX-RS** – API dla RESTful Web Services
- **EL** – *Expression Language* – mechanizm pozwalający na komunikację warstwy webowej z logiką aplikacji
- **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 procesu działania aplikacji
- **CDI** – *Context and Dependency Injection* – mechanizm wstrzykiwania zależności
- **Managed Beans** – specjalne klasy pozwalające na dostęp oraz manipulację danymi – rozdzielają warstwę prezentacji i logiki w JSF
- **EJB** – *Enterprise JavaBeans* – serwerowe klasy pozwalające na implementację logiki biznesowej

# Java Enterprise Edition Model

- **JPA** – *Java Persistence API* – definiuje standard interfejsów opisujących warstwę persystencji aplikacji (obsługa baz danych)
- **JTA** – *Java Transaction API* – definiuje standard interfejsów opisujących użycie transakcji w aplikacji
- **JMS** – *Java Messaging Service* – zestaw interfejsów asynchronicznego przesyłania komunikatów
- **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.

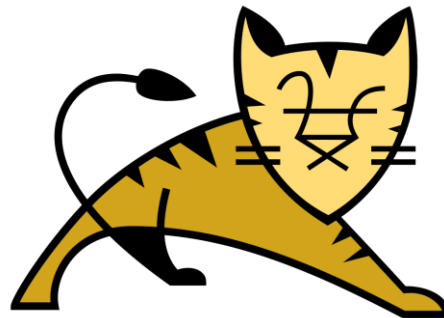
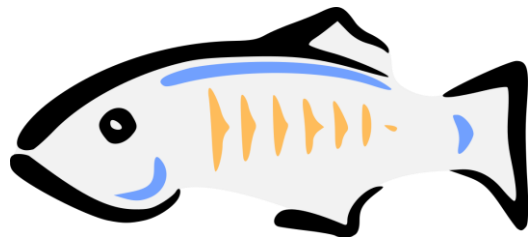
JEE jest rozszerzeniem dla JSE. Dostarcza **API** oraz środowisko uruchomieniowe dla aplikacji budowanych na wielką skalę, wielowarstwowych, skalowalnych, *niezawodnych*.

2.

# Aplikacje JEE na start!

Serwery JEE

# Szeroki wachlarz serwerów JEE



WildFly



WebSphere



# Serwer JEE

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

Nasz wybór:



# Zadanie 2.1: 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 2.2: 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 /home/<user>/.bash_profile
export JBOSS_HOME=/home/<user>/wildfly
export WILDFLY_HOME=$JBOSS_HOME
```

```
$ source ~/.bash_profile
```

**Do pliku .bashrc należy dodać:**

```
. /home/<user>/.bash_profile
```

## Zadanie 2.3: Nowy użytkownik Wildfly

```
# cd $WILDFLY_HOME  
# ./bin/add-user.sh
```

Dokonujemy wyboru (a) Management User

Nadajemy własną nazwę i hasło.

Zezwalamy na dostęp do zdalnego API.

I gotowe!

## Zadanie 2.4: 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 oraz zbudowanie docelowego artefaktu typu jar lub war.

To, do jakiego pliku ostatecznie nasza aplikacja zostanie zapakowana, określa konfiguracja w pom.xml:

```
<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 kontenerze 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 3.1

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

```
<groupId>com.isa</groupId>  
<artifactId>users-engine</artifactId>  
<version>1.0-SNAPSHOT</version>
```



## Zadanie 3.2.1

- Utwórz pakiet **com.isa.usersengine.domain**
- Stwórz w nim klasę **User** z polami: **id**, **name**, **login**, **password**, **age**.
- Zapewnij **getter**y i **setter**y dla wskazanych pól

## Zadanie 3.2.2

- Utwórz pakiet **com.isa.usersengine.dao**
- Stwórz w nim interfejs **UsersRepositoryDao** z metodami: **addUser**, **getUserById**, **getUserByLogin**, **getUsersList** – jak powinny wyglądać sygnatury metod oraz jaki powinien być typ zwracany przez te metody?
- Dostarczone repozytorium **UsersRepository** umieść w pakiecie **com.isa.usersengine.repository**
- W pakiecie **com.isa.usersengine.dao** utwórz klasę **UsersRepositoryDaoBean** implementującą interfejs **UsersRepositoryDao**. Zaimplementuj wymagane metody wykorzystując klasę **com.isa.usersengine.repository.UsersRepository**

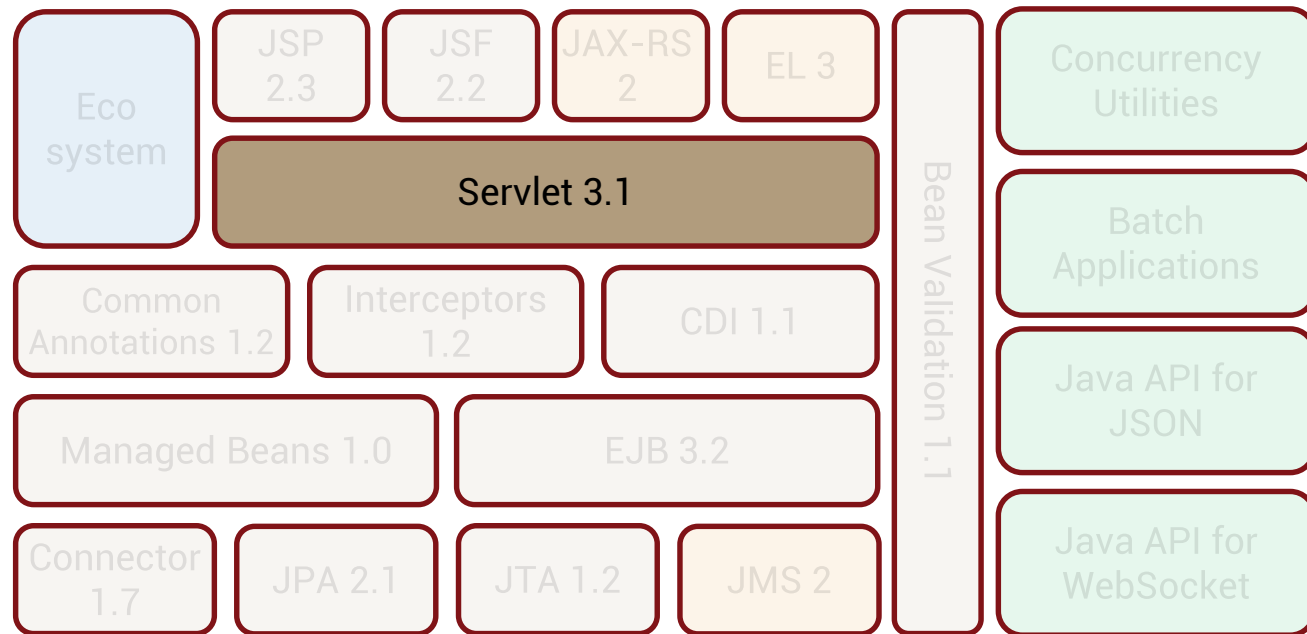
## Zadanie 3.2.3

- Wykorzystaj klasę **Main** do wyświetlenia imion wszystkich użytkowników repozytorium. Użyj **DAO**.
- Dlaczego używamy klasy **UsersRepository**?

# 4. Servlet

Request-Response Communication

# JEE 7



# 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.

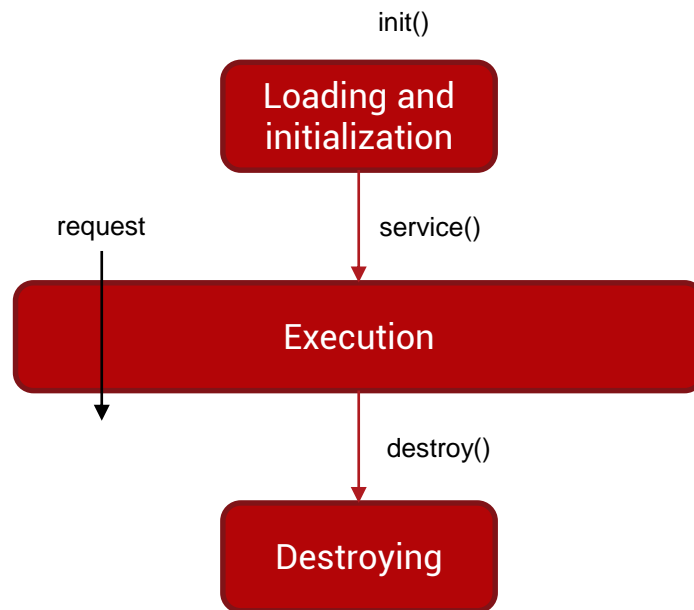
Z oparcia o nasz wybór, w **Wildfly** kontenerem webowym jest **Undertow**, którego konfigurację możemy znaleźć na liście subsystemów serwera.

# Zależności do pracy z Servletem

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



# Servlet Lifecycle



# Metody komunikacji

## HTTP

**GET** – odczyt rekordu

**POST** – tworzenie rekordu

**PUT** – edycja rekordu

**DELETE** – kasowanie rekordu

# Rozpoczęcie pracy z Servletem – obsługa GET

```
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
    }
}
```

# Servlet

## Obsługa requestu

Servlet jest wrażliwy na zadane parametry requestu.

`http://[host]:[port]/[servlet-context]?name=John&age=32`

Zmienna lokalna **HttpServletRequest#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

## Interpretacja GET

- Wszystkie parametry wysłane na przykład przez adres URL w przeglądarce znajdują się w obiekcie **requestu** i dostępne są przez metodę **getParameter(String var1)**
- Metoda **getParameter(String var1)** zwraca obiekt typu **String**. Należy dokonać rzutowania/parsowania do oczekiwanego typu na własną rękę.
- **UWAGA!** Parametry **requestu** są typu **read-only**. Nie ma możliwości zmiany ich wartości.

# Servlet

## Obsługa odpowiedzi

Za pomocą servletu również możemy generować odpowiedzi.

Zmienna lokalna **HttpServletResponse#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 4.1

- Stwórz pakiet **com.isa.usersengine.servlets** – tutaj umieszczaj wszystkie kolejne servlety
- Stwórz pierwszy servlet o nazwie **HelloServlet** w kontekście **hello-servlet**
- Spraw aby wyświetlał on **Hello World from my first Servlet!**
- 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 Namespace

Domyślnym adresem naszej aplikacji jest:

`http://[host]:[port]/${project.artifactId}/[servlet-context]`

Istnieje możliwość nadania własnej ścieżki do aplikacji. Zamiast zmiennej `${project.artifactId}` możemy użyć dowolnego ciągu znaków.

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://[host]:[port]/${project.artifactId}/[servlet-context]`

Odwołamy się:

`http://[host]:[port]/[servlet-context]`

W tym celu definiujemy plik **jboss-web.xml** i umieszczamy go w katalogu **webapp/WEB-INF**

# 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 4.2

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

## Zadanie 4.3

- 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 requesce, zwróć status **BAD\_REQUEST** – wykorzystaj do tego klasę ze statycznymi kodami **HttpServletResponse**

## Zadanie 4.4

- Utwórz nowy servlet **FindUserByIdServlet** w kontekście **find-user-by-id**
- Wykonaj wyszukiwanie użytkownika po zadanym w request parametrze **id**.
- Jeśli parametr **id** nie został podany w request, zwróć status **BAD\_REQUEST** – wykorzystaj do tego klasę ze statycznymi kodami **HttpServletResponse**
- Do rozwiązania wykorzystaj klasy **DAO**, **domain**, **repository**

# Rozpoczęcie pracy z Servletem – 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("/add-user")
public class AddUserServlet extends HttpServlet {

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

        //         provide your code here
    }
}
```

# Servlet

## Interpretacja POST

- Interpretacja komunikacji typu POST jest analogiczna do typu GET.
- Wszystkie parametry wysłane formularzem znajdują się w obiekcie **requestu** dostępne przed metodę **getParameter(String var1)**
- Metoda **getParameter(String var1)** zwraca obiekt typu **String**. Należy dokonać rzutowania/parsowania do oczekiwanego typu na własną rękę.
- **UWAGA!** Parametry **requestu** są typu **read-only**. Nie ma możliwości zmiany ich wartości.

# Servlet

## Parametry & Atrybuty – request scoped

- **UWAGA!** Parametry **requestu** są typu **read-only**. Nie ma możliwości zmiany ich wartości.
- Aby zapisać nowe wartości parametrów w **request** i przekazać je dalej w aplikacji webowej, np. do innego servletu lub widoku korzystamy z atrybutów.

```
String price = req.getParameter("price");
```

```
req.setAttribute("doubledPrice", Integer.parseInt(price) * 2);
```

```
int doubledPrice = (int) req.getAttribute("doubledPrice");
```



# Servlet

## Parametry & Atrybuty – session scoped

- **UWAGA!** Parametry **requestu** są typu **read-only**. Nie ma możliwości zmiany ich wartości.
- Aby zapisać nowe wartości parametrów w **sesji** i przekazać je dalej w aplikacji webowej, np. do innego servletu lub widoku korzystamy z atrybutów.

```
String price = req.getParameter("price");
```

```
req.getSession().setAttribute("doubledPrice", Integer.parseInt(price) * 2);
```

```
int doubledPrice = (int) req.getSession().getAttribute("doubledPrice");
```

## Zadanie 4.5

- Utwórz nowy Servlet **AddUserServlet**, który będzie obsługiwał metodę komunikacji **POST**
- Użyj dostarczonego pliku **add-user.html** do dodawania użytkownika
- Wykonaj dodawanie nowego użytkownika wg danych podanych w formularzu do repozytorium użytkowników w pamięci

# Servlet & Wildfly

## Śledzenie deploymentu

Na potrzeby debugowania, śledzenia zmian, pozyskania informacji o zdeployowanej aplikacji, testów wydajności, itp... istnieje mechanizm informujący o zarejestrowanych w kontenerze aplikacji servletach oraz przechowujący statystyki użycia poszczególnych servletów.

Informacje o zarejestrowanych servletach (w tym o statystykach):

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

# Servlet & Wildfly

## Śledzenie deploymentu – aktywacja statystyk

Aby statystyki były zbierane, należy je aktywować:

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

## Zadanie 4.6: Servlet Statystyki

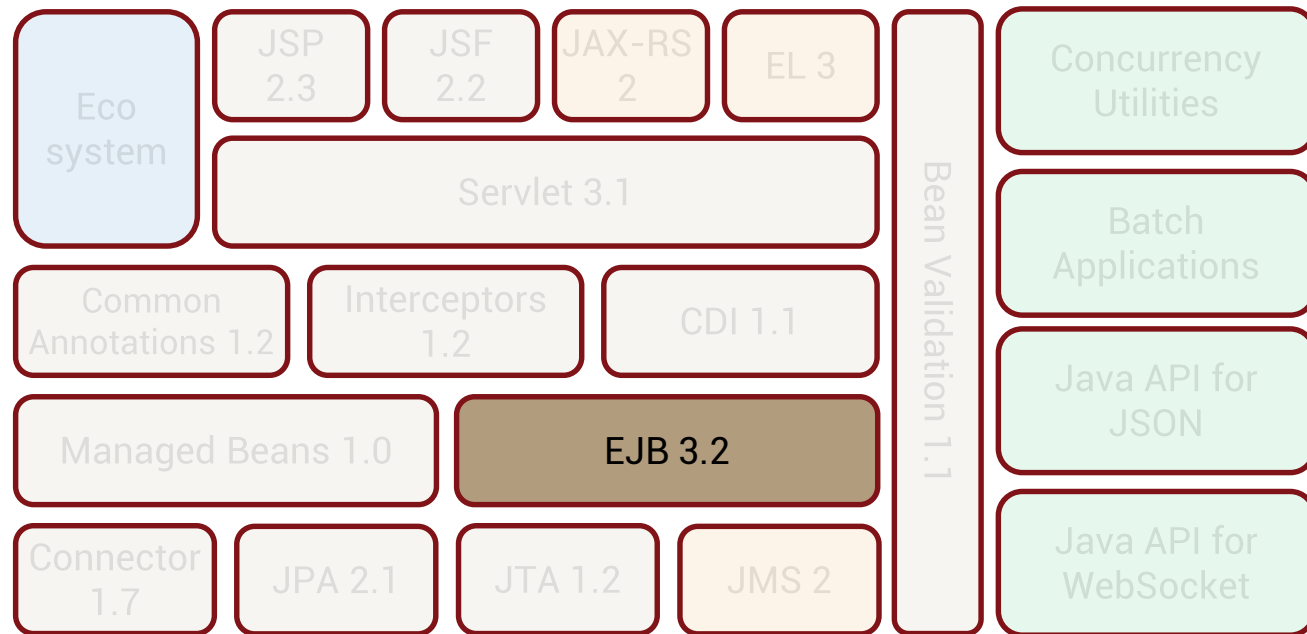
- Aktywuj statystyki servletów
- Przetestuj ich działanie

5.

## EJB / CDI

Enterprise Java Bean / Context and Dependency Injection

# JEE 7

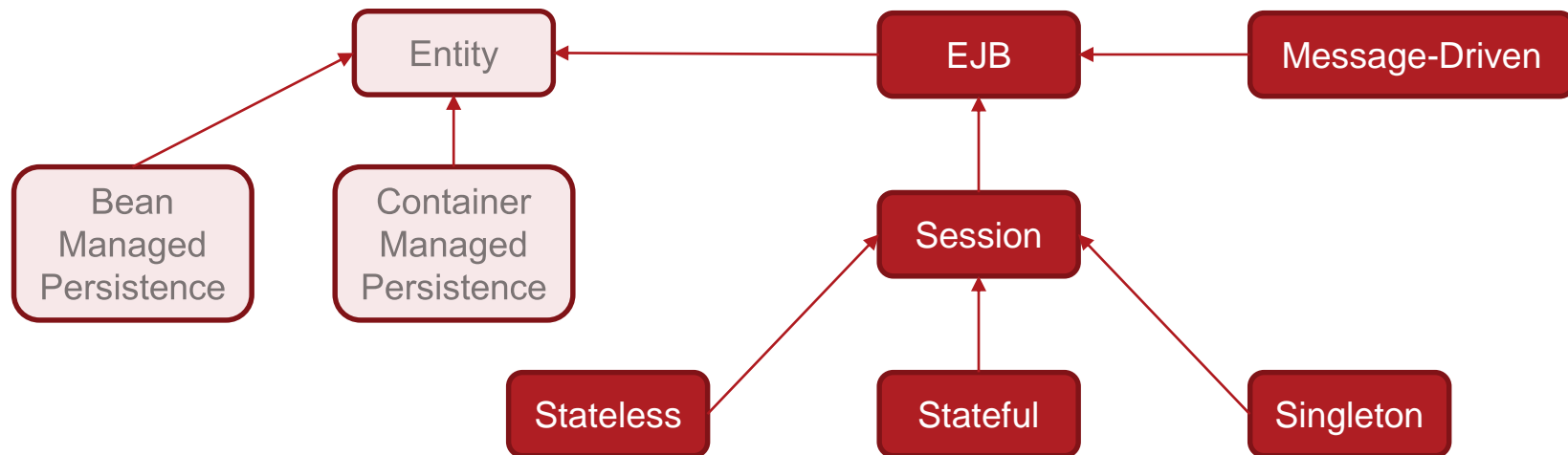


# Czym jest EJB ?

- 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

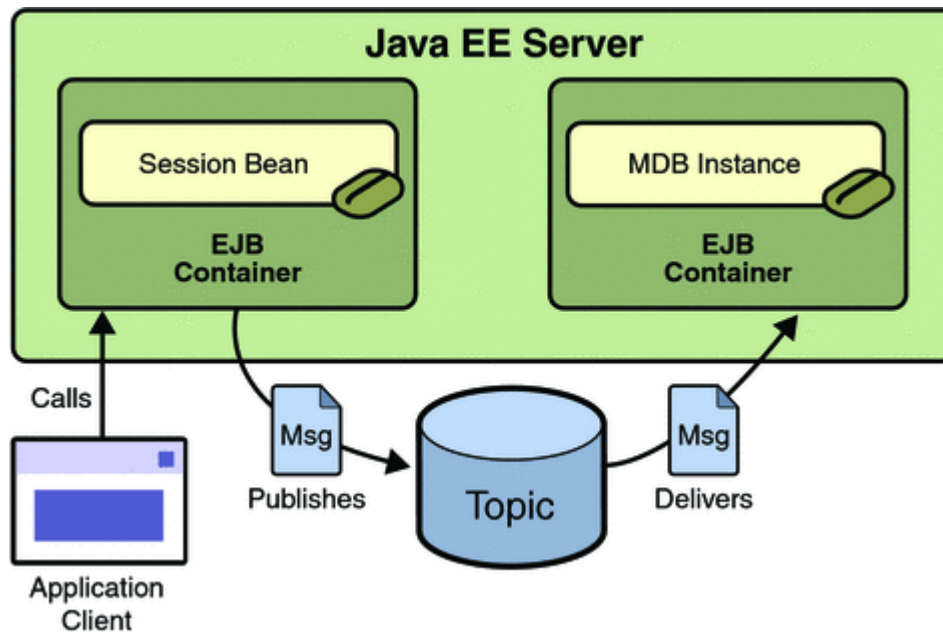
Komponent sterowany wiadomościami, często opierający się na mechanizmie JMS (część specyfikacji JEE).

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 5.1

- Przekonwertuj interfejs **UsersRepositoryDao** oraz klasę **UsersRepositoryDaoBean** na **bezstanowe EJB** o zakresie **lokalnym**.

# EJB

## Dependency Injection

- Użycie zdefiniowanych EJB wykorzystuje adnotacje **@EJB/@Inject**
- **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 5.2

- Przekształć servlet **FindUserByIdServlet** na postać korzystającą z EJB DI.

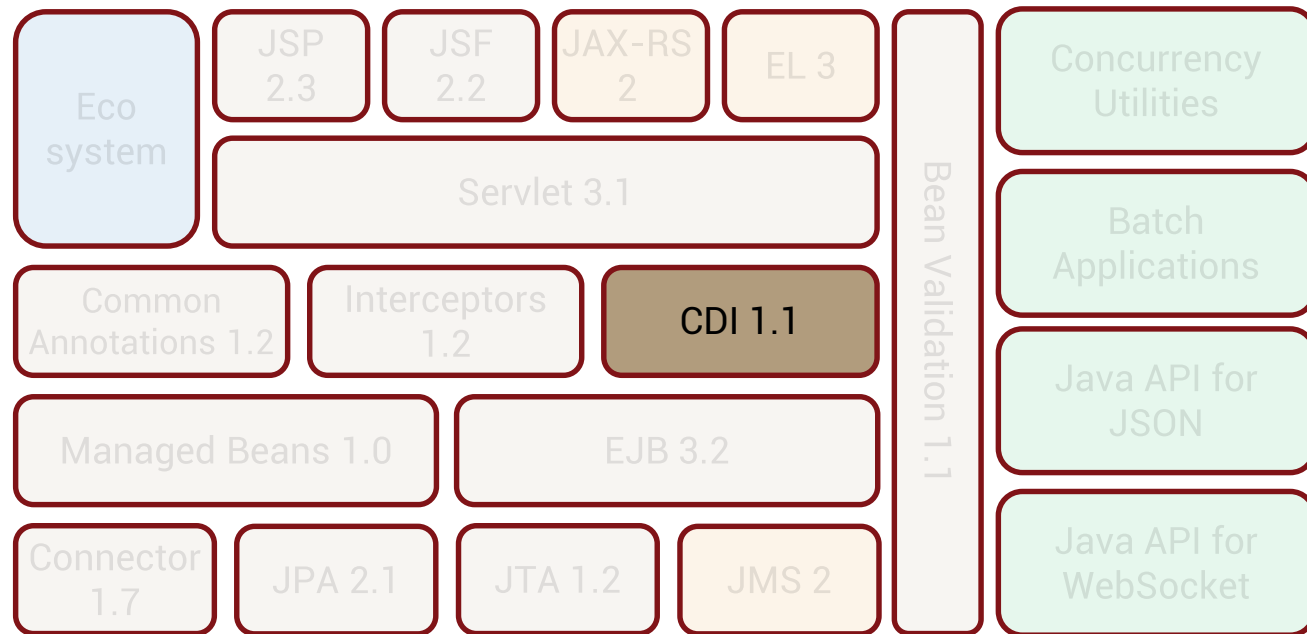
# CDI

## Context and Dependency Injection

Zestaw usług pozwalający na zachowanie **loose coupling** między warstwami aplikacji.

Pozwala na wstrzyknięcie większości obiektów występujących w ramach aplikacji. Nie muszą być one **EJB**.

# JEE 7





# CDI

## Zakresy działania

Oznaczenia beanów **CDI**:

**@ApplicationScoped** – stan współdzielony przez użytkowników w kontekście całej aplikacji

**@SessionScoped** – stan na czas interakcji użytkownika z aplikacją webową w ramach wielu requestów

**@RequestScoped** – stan na czas interakcji użytkownika z aplikacją webową w ramach jednego requestu

## Zadanie 5.3

- Stwórz pakiet **com.isa.usersengine.cdi**
- Stwórz w nim trzy interfejsy: **RandomUserCDIApplicationDao**, **RandomUserCDIRequestDao**, **RandomUserCDISessionDao**
- Każdy z interfejsów powinien definiować jedną metodę: **User getRandomUser();**
- Stwórz trzy implementacje tych interfejsów w postaci CDI beanów o zakresie request, session, application (analogicznie względem nazw). Metoda powinna zwracać losowego użytkownika z repozytorium.
- Utwórz servlet **RandomUserServlet** i wyświetl w nim wynik metody pochodzącej z każdego z trzech powyższych beanów.
- Przeprowadź eksperyment wyświetlając i odświeżając stronę w przeglądarce (karta standardowa oraz karta incognito)

# CDI

## Oznaczenia

Oznaczenia beanów CDI:

**@Interceptor** – uczestniczy w procesie działania aplikacji dostarczając dodatkową logikę, odseparowaną od logiki biznesowej aplikacji

**@Decorator** – możliwość dodawania nowej logiki dla obiektów dekorowanych klas

**@Stereotype** – wiązanie adnotacji w celach re-używalności definiowanych zachowań/cech

# 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

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

```
@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 5.4

- Stwórz nowy typ **enum Gender {MAN, WOMAN}** w pakiecie **com.isa.usersengine.domain**
- Dodaj atrybut płeć (**gender**) do klasy **User**. Zapewnij obsługę tego atrybutu. Uzupełnij repozytorium użytkowników o wartość tego atrybutu.
- Utwórz nowy CDI Bean o nazwie **MaxPulseBean** o zakresie requestu, a w nim dwie metody liczące maksymalny statystyczny puls dla kobiet i dla mężczyzn.
- Rozszerz funkcjonalność servletu **FindUserByIdServlet** o wyświetlenie odpowiedniej wartości.

$MEN = 202 - (0.55 * age);$

$WOMEN = 216 - (1.09 * age);$

## Zadanie 5.5

- Zaimplementuj pozostałe metody klasy **UsersRepositoryDaoBean** jeśli nadal nie istnieją.





# Thanks!!

Q?