


# Podstawy Java Enterprise Edition



# Cześć

Łukasz Chrzanowski

 /lukasz-chrzanowski-dev

 /lukasz4coders

[lukasz@chrzanowski.co](mailto:lukasz@chrzanowski.co)

# Materiały do zajęć

<https://github.com/infoshareacademy/jjddr1-materialy-jee>

# Parameter vs Attribute

# Request: Parameter vs Attribute

**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");
```

# Session: Parameter vs Attribute

**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");
```

# Session

Sesja tworzona jest przez serwer przy pierwszym odwołaniu się do niej

- `req.getSession(boolean create)` ;
- `create` - czy sesja powinna być utworzona, gdy jeszcze nie istnieje?

Po wywołaniu metody **request.getSession()** do odpowiedzi serwera dołączane będzie ciasteczko, czyli fragment tekstu z unikalnym identyfikatorem.

Ciasteczka identyfikujące sesję użytkownika w Javie EE mają nazwę **JSESSIONID**

# Zadanie: Sesja



- Przygotuj servlet **LoginServlet** w kontekście **login**
- Utwórz nowy widok **login.jsp**, wykorzystaj plik **add-user.html**
- W nowo utworzonym widoku, przygotuj formularz z polami login i password
- W servlecie **LoginServlet**, odbierz parametry z formularza, sprawdź czy  
login == "isa@isa.pl"  
password == "123456"
- Zachowaj login w sesji
- Przygotuj servlet **AdminServlet** w kontekście **admin**
- Sprawdź w servlecie czy sesja posiada atrybut login, jeżeli tak wyświetl komunikat "Panel Administracyjny", w przeciwnym wypadku ustaw status na 403 - Forbidden



# Session - Timeout

Ustawienie określonego czasu trwania sesji:

- `session.setMaxInactiveInterval(timeout);`
- `timeout` - czasem podanym w sekundach

# Śledzenie deploymentu

# 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 -> Management model -> subsystem -> undertow -> servlet -> [:servlet]**

# Zbieranie statystyk

Aby statystyki były zbierane, należy je aktywować:

**Configuration -> Subsystems -> Web – Undertow -> Global Settings  
-> View -> Edit -> **Statistics enabled=true****

Pamiętaj o restarcie serwera!

# Zadanie: Statystyki

Aktywuj statystyki servletów

Przetestuj ich działanie

# EJB / CDI

Enterprise Java Bean / Context and Dependency Injection

# EJB: Charakterystyka

- 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

# EJB: Charakterystyka

Jedyną rzeczą wymaganą od programisty korzystającego ze specyfikacji EJB jest dostosowanie się do pewnego interfejsu EJB (**wymogów implementacyjnych**), którego zastosowanie zwalnia użytkownika EJB z konieczności opracowywania własnych metod obsługi komponentów.



# EJB: Charakterystyka

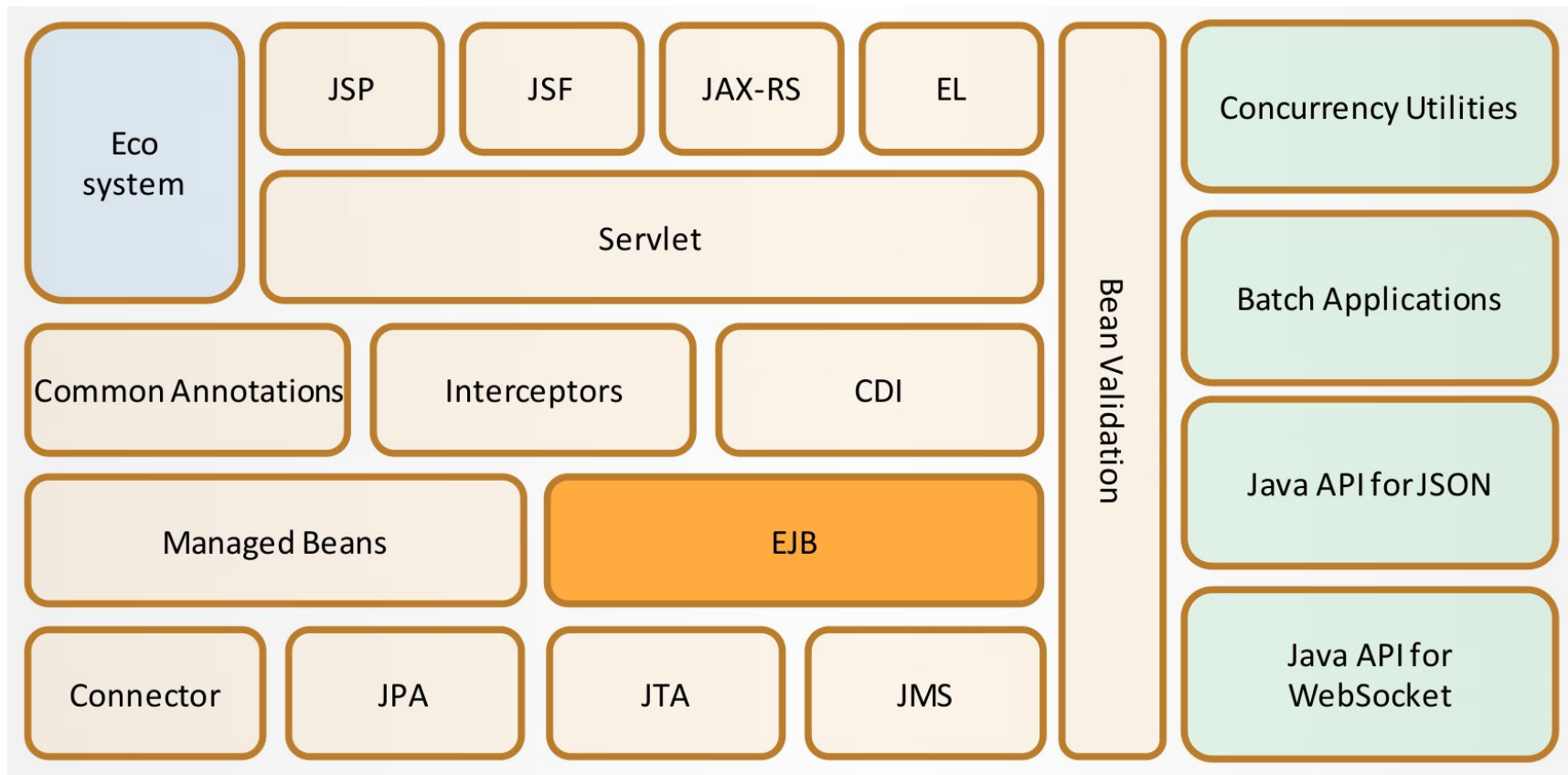
Idea **EJB** opiera się na tworzeniu komponentów (**ziaren EJB**), które mogą być osadzone na serwerze aplikacji (tzw. kontenerze EJB), który z kolei udostępnia je do wykonania lokalnie (dostęp z części aplikacji uruchomionej na tej samej wirtualnej maszynie) lub zdalnie poprzez protokół RMI.

# EJB: Charakterystyka

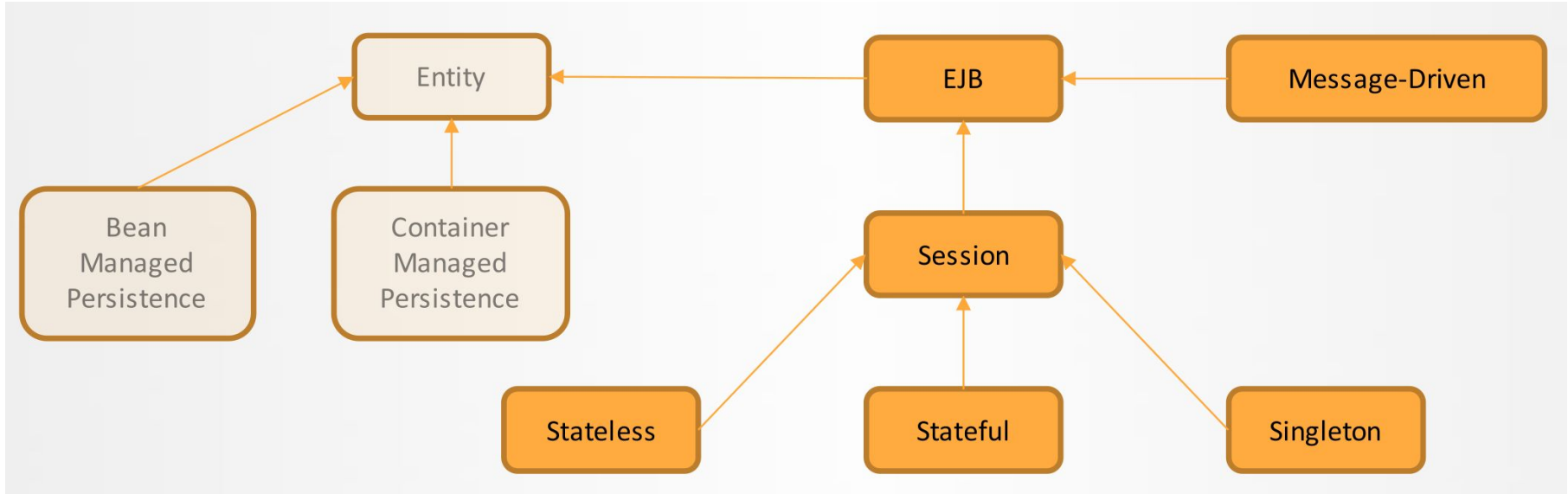
Główną zaletą **EJB** jest nakierowanie projektanta na pewne sprawdzone sposoby rozwiązywania typowych problemów w systemie rozproszonym:

- zarządzanie połączeniami,
- transakcja rozproszona,
- mapowanie danych na model obiektowy itp.

# Komponenty



# EJB: Budowa



# Message-Driven: Charakterystyka

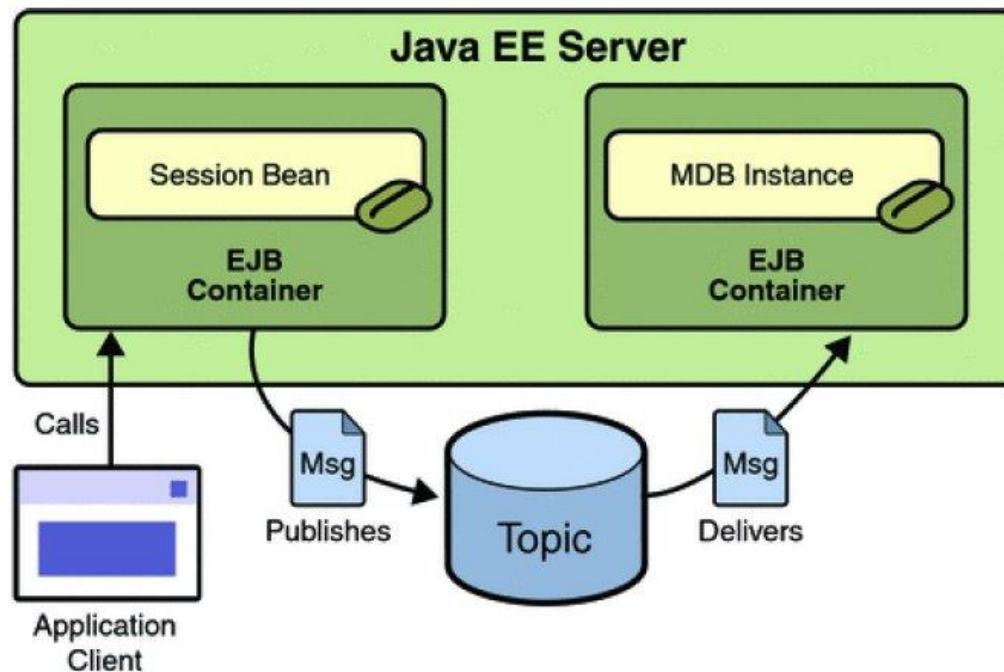
Komponent sterowany wiadomościami, często opierający się na mechanizmie JMS (część specyfikacji JEE).

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

Taka obsługa pozwala na podejście całkowicie asynchroniczne.

Nie posiada żadnych informacji o systemie, który przetwarza te informacje wie tylko gdzie się znajduje. W EJB który działa poprzez RMI czyli zdalne wywoływanie metod, podaje się interfejsy, które są niezbędne do wywoływania metod biznesowych.

# MDB: Schemat działania



źródło: oracle.com

# Session Bean: Charakterystyka

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 Bean: 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



# Przykład: Stanowość

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

```
@Stateless
```

```
public class UsersRepositoryDaoBean implements UsersRepositoryDao  
{  
    ...  
}
```

# Stanowość - Stateless

Metoda z adnotacją **@Remove** - Po wywołaniu takiej metody jest to sygnał dla kontenera, aby usunąć bezstanowe ziarno ze swoich zasobów, a w samej metodzie możemy dokonać "sprzątania" takiego jak np. zamknięcie połączeń z bazą danych, czy zwrócenie ich do puli.

W przypadku, gdy nasze ziarno korzysta z dodatkowych zasobów jak np. połączenie z bazą danych, to warto dodać także dwie metody powiązane z cyklem życia. Zwolnienie zasobu przechowywanego przez obiekt ziarna możemy zrobić w metodzie oznaczonej adnotacją **@PrePassivate**, natomiast przy ponownej aktywacji pobrać je z puli ponownie w metodzie z adnotacją **@PostActivate**.

# Session Bean: Zasięg

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

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.

# Przykład: Zasięg

```
import javax.ejb.Local;  
  
@Local  
public interface UsersRepositoryDao {  
  
    ...  
  
}
```

# Zadanie: EJB

Przekonwertuj interfejs **UsersRepository** oraz klasę **UsersRepositoryBean** na **bezstanowe EJB** o zakresie **lokalnym**.

# EJB: Wstrzykiwanie zależności

- 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).

# 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**. **EJB** jest kontenerem **DI**.



# Przykład: Dependency injection

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

Wstrzykujemy zależność wykorzystując interfejs.

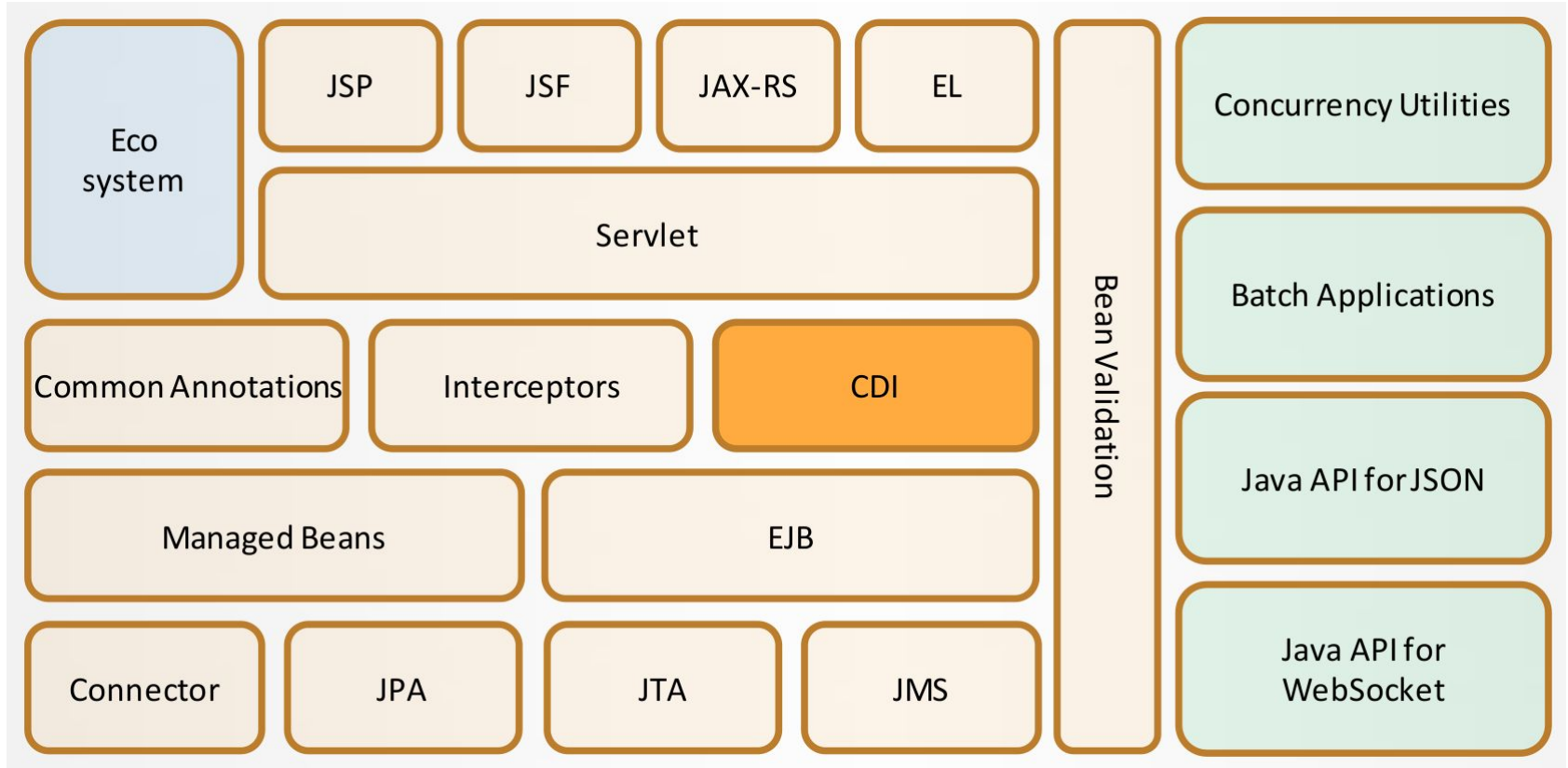
```
public class User {  
  
    @EJB  
    PermissionsInterface permissions;  
  
    public User() {  
    }  
}
```

# CDI: Context and Dependency Inj.

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

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

# Komponenty



# CDI: Charakterystyka

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

# EJB > CDI

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

# @EJB vs @Inject

**@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.

## Zadanie: CDI



- 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** w kontekście webowym **/random-user** 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)

## Zadanie: CDI



- 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 equestu, 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: JEE Elementary



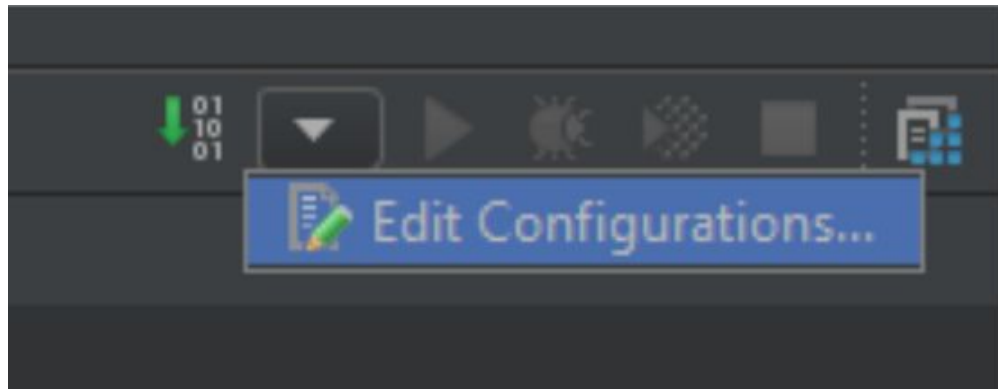
Zaimplementuj pozostałe metody klasy **UsersRepositoryBean** jeśli nadal nie istnieją.

# Debugger

# Konfiguracja wbudowana

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 Wildfly dla projektu

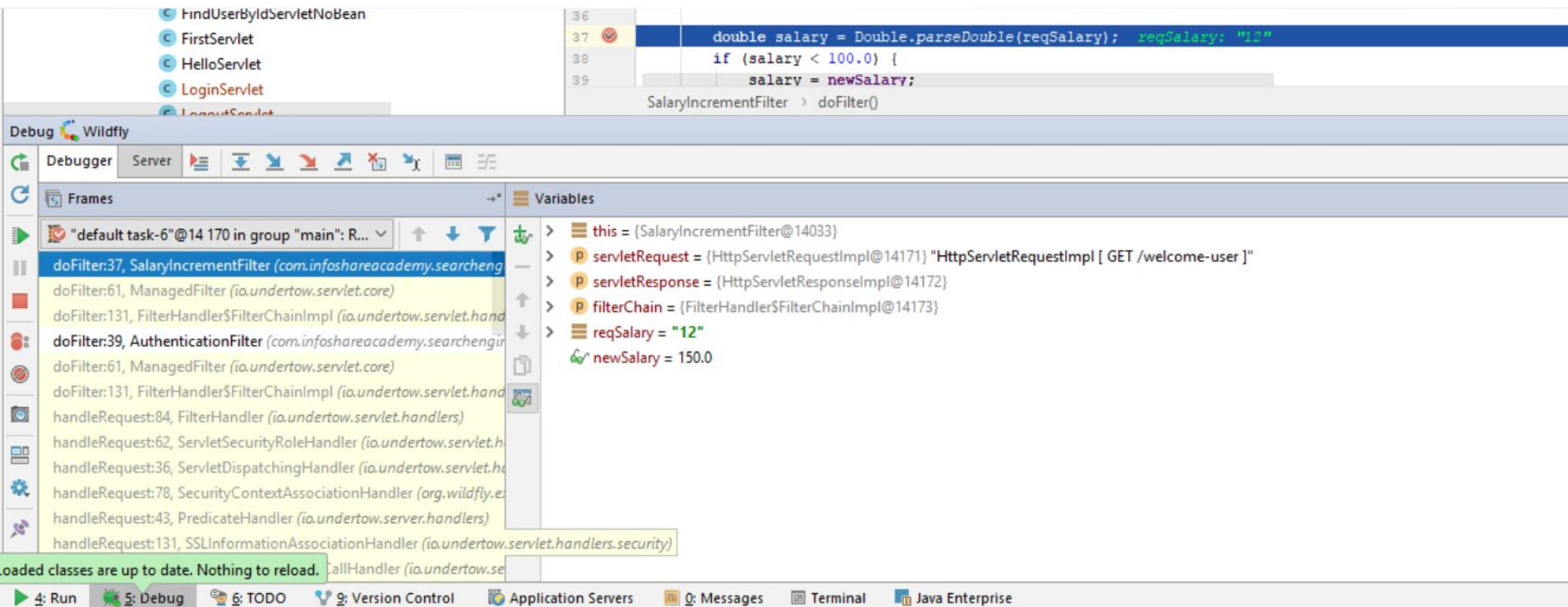
W pozycji **Application Server** wybieramy **Configure**.  
Jako **Jboss Home** wskazujemy główny folder **wildfly**.

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

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

Dodatkowo możemy nadać nazwę naszej konfiguracji.

# Wstęp do debuggera



The screenshot displays an IDE interface for debugging a Java application. The top panel shows the source code of the `SalaryIncrementFilter.doFilter()` method. The bottom panel shows the Debug Console with a list of frames and variables.

**Source Code:**

```
36  
37 double salary = Double.parseDouble(reqSalary); reqSalary: "12"  
38 if (salary < 100.0) {  
39     salary = newSalary;  
SalaryIncrementFilter > doFilter()
```

**Debug Console:**

Debugger: Wildfly

Frames:

- "default task-6" @ 14 170 in group "main": R...
- doFilter:37, SalaryIncrementFilter (com.infoshareacademy.searcheng...)
- doFilter:61, ManagedFilter (io.undertow.servlet.core)
- doFilter:131, FilterHandler\$FilterChainImpl (io.undertow.servlet.han...
- doFilter:39, AuthenticationFilter (com.infoshareacademy.searchengir...)
- doFilter:61, ManagedFilter (io.undertow.servlet.core)
- doFilter:131, FilterHandler\$FilterChainImpl (io.undertow.servlet.han...
- handleRequest:84, FilterHandler (io.undertow.servlet.handlers)
- handleRequest:62, ServletSecurityRoleHandler (io.undertow.servlet.h...
- handleRequest:36, ServletDispatchingHandler (io.undertow.servlet.h...
- handleRequest:78, SecurityContextAssociationHandler (org.wildfly.e...
- handleRequest:43, PredicateHandler (io.undertow.server.handlers)
- handleRequest:131, SSLInformationAssociationHandler (io.undertow.servlet.handlers.security)

Variables:

- this = {SalaryIncrementFilter@14033}
- HttpServletRequest = {HttpServletRequestImpl@14171} "HttpServletRequest [ GET /welcome-user ]"
- ServletResponse = {HttpServletResponseImpl@14172}
- filterChain = {FilterHandler\$FilterChainImpl@14173}
- reqSalary = "12"
- newSalary = 150.0

loaded classes are up to date. Nothing to reload. [allHandler (io.undertow.se...

4: Run 5: Debug 6: TODO 7: Version Control 8: Application Servers 9: Messages 10: Terminal 11: Java Enterprise

# Konfiguracja Wildfly dla projektu

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**

Dodatkowo możemy nadać nazwę naszej konfiguracji.



# Dzięki