Podstawy JEE

Dominika Makuch

Cloud Software Engineer @ Intel

1.
JEE - Java Enterprise Edition

Java Enterprise Edition

Co to jest?

- “Enterprise” - do wykorzystania w aplikacjach przemysłowych
- Zestaw specyfikacji API dla języka Java, które mają upraszczać i przyspieszać wytwarzanie przemysłowego oprogramowania

J2EE -> JEE
historia

J2EE 1.2 1999

J2EE 1.3 2001

J2EE 1.4 2003

JEE 5 2006

JEE 6 2009

JEE 7 2013

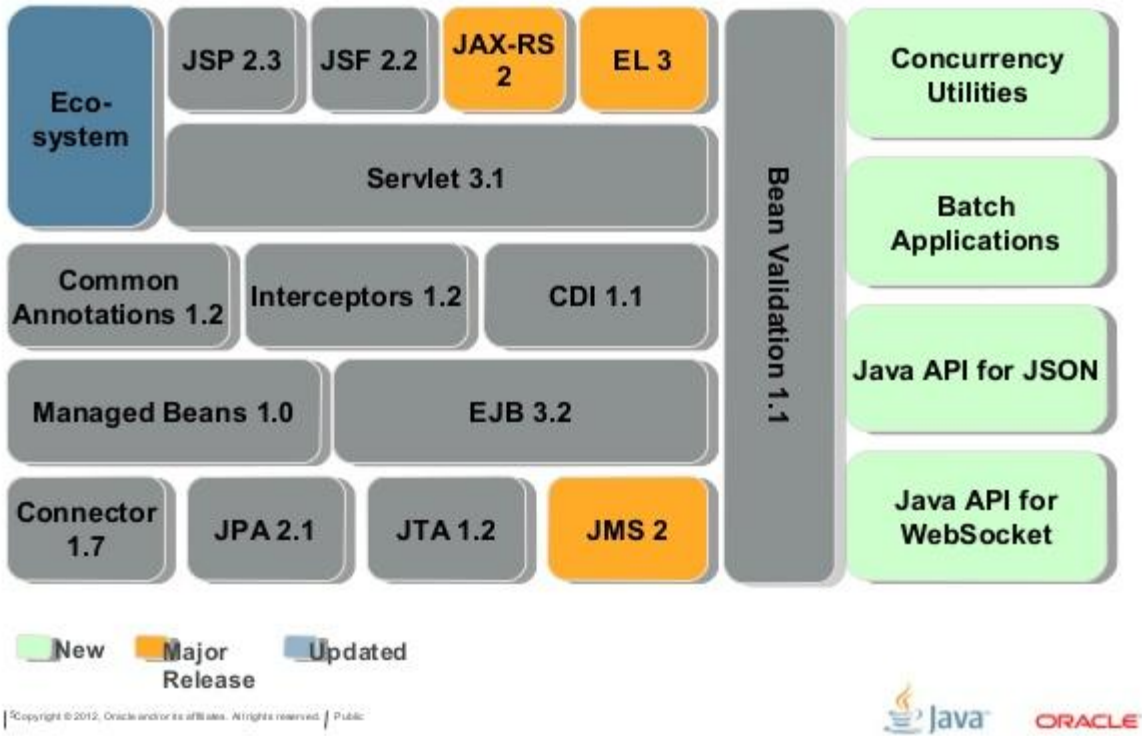
JEE 8 2017

The Java EE Platform specification is the umbrella specification that defines the Java EE platform. The platform specification doesn't define the Java EE APIs directly, but rather includes them by reference to other Java specifications and defines how they all fit together in the overall Java EE platform. The platform specification also defines other attributes of the platform such as security, deployment, transactions, and interoperability.

https://javaee.github.io/javaee-spec/

JEE 7

Java EE 7



Java ORACLE

2.
Serwer JEE

Serwery JEE

Aplikacja JEE jest uruchomiona na serwerze JEE, który odpowiada za bezpieczeństwo, skalowalność i zarządzanie jej cyklem życia

Zadanie 1
Aplikacja JSE

- Stwórz nowy projekt (Maven)
- Zaimplementuj klasę NumbersGenerator z metodą

```
public int getRandomInt(int upperBound)
```

- Dopisz klasę App z metodą `main()` która wypisuje na konsolę losowy int
- Uruchom z IDE

Zadanie 2
Aplikacja JSE - uruchamialny JAR

- Utwórz uruchamialnego JARa ze swojej aplikacji
  - maven-jar-plugin
- Uruchom aplikację z konsoli

Zadanie 2
Aplikacja JSE - uruchamialny JAR

- Utwórz uruchamialnego JARa ze swojej aplikacji
  - maven-jar-plugin
- Uruchom aplikację z konsoli

- W następnych zadaniach zmienimy ją w usługę internetową serwującą losowe liczby

Serwer JEE
Wildfly

- Rozwijany przez RedHat
- Przed 2014 nazywał się JBoss

Zadanie 3
Instalacja Wildfly - przygotowanie plików

1. Ściągamy paczkę z Wildfly:
   http://download.jboss.org/wildfly/10.1.0.Final/wildfly-10.1.0.Final.zip
2. Rozpakowujemy do katalogu /opt/:
   sudo unzip wildfly-10.1.0.Final.zip -d /opt
3. Tworzymy link symboliczny do katalogu serwera:
   sudo ln -s wildfly-10.1.0.Final/ /opt/wildfly
4. Dodajemy w systemie usera, z którego będzie uruchamiany serwer:
   adduser --no-create-home --disabled-password --disabled-login wildfly
5. Zmieniamy uprawnienia do katalogu serwera:
   sudo chown -R wildfly:wildfly wildfly*

Zadanie 3

Instalacja Wildfly - rejestracja jako Ubuntu service

6. Kopiujemy skrypt startowy serwera we właściwe dla Ubuntu miejsce:
   sudo cp docs/contrib/scripts/init.d/wildfly-init-debian.sh /etc/init.d/wildfly
7. Instalujemy dowiązanie do skryptu startowego:
   sudo update-rc.d wildfly defaults
8. Kopiujemy plik konfiguracyjny serwera we właściwe miejsce:
   sudo cp /opt/wildfly/docs/contrib/scripts/init.d/wildfly.conf /etc/default/wildfly

Zadanie 3
Instalacja Wildfly - konfiguracja serwera

9. Otwieramy plik konfiguracyjny:
   sudo vim /etc/default/wildfly
10. Ustawiamy (odkomentowujemy) następujące parametry:

```
## Location of JDK
 JAVA_HOME="/usr/lib/jvm/default-java"

## Location of WildFly
 JBOSS_HOME="/opt/wildfly"

## The username who should own the process.
 JBOSS_USER=wildfly

## The mode WildFly should start, standalone or domain
 JBOSS_MODE=standalone
```

```
## Configuration for standalone mode
 JBOSS_CONFIG=standalone-full.xml

## The amount of time to wait for startup
 STARTUP_WAIT=60

## The amount of time to wait for shutdown
 SHUTDOWN_WAIT=60

## Location to keep the console log
 JBOSS_CONSOLE_LOG="/var/log/wildfly/console.log"
```

Zadanie 3
Instalacja Wildfly - start serwera

11. Uruchamiamy serwer:

```
service wildfly start
```

12. Zatrzymujemy serwer:

```
service wildfly stop
```

Zadanie 3
Serwer Wildfly - logi, konsola administracyjna

- Logi serwera znajdują się w katalogu: /var/log/wildfly/console.log
- Obserwacja logów:

```
tail -f /var/log/wildfly/console.log
```

- Zatrzymaj serwer i wystartuj go ponownie - obserwuj logi

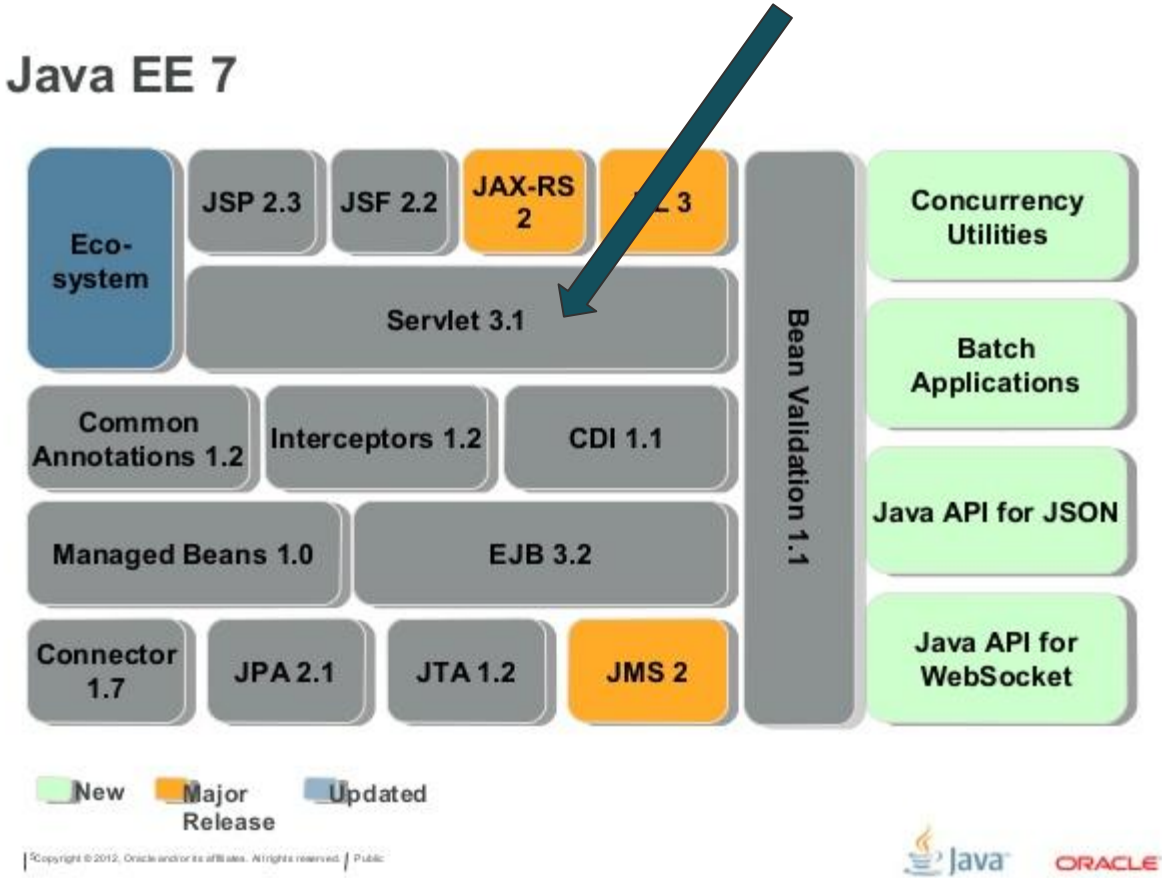
Zadanie 3
Serwer Wildfly - logi, konsola administracyjna

- Otwórz przeglądarkę i przejdź do adresu *localhost:8080*
  - Co oznacza *localhost:8080?*

- Przejdź do konsoli administracyjnej
- Ustaw usera i hasło do konsoli administracyjnej serwera:
  - cd /opt/wildfly/bin/
  - sudo ./add-user.sh

3.
Servlet API

JEE 7


Java EE 7
Eco-system
JSP 2.3
JSF 2.2
JAX-RS 2
L 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
New
Major Release
Updated
Copyright © 2012, Oracle and/or its affiliates. All rights reserved. | Public
Java
ORACLE

Java Servlet API

- Specyfikacja dla klas, które odpowiadają za przetwarzanie requestów Http (Http Servlet)
- Servletami zarządza część serwera JEE zwana kontenerem webowym
- Kontener webowy w serwerze Wildfly 10 nazywa się Undertow

Java Servlet API
Zadanie

- Otwórz konsolę administracyjną serwera Wildfly i znajdź podsystem Undertow

Java Servlet API

```
@WebServlet("/url-path")
public class SimpleServlet extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
        // code here
   }
}
```

Zadanie 4
Hello World Servlet

- Dołącz do projektu zależność javaee-api
- Ustaw packaging na *war*
- Dodaj konfigurację wtyczki maven-war-plugin
- Napisz servlet wyświetlający “Hello World”

Zadanie 4
Hello World Servlet

- Zbuduj projekt
- Wykonaj deployment na serwer Wildfly

  `sudo cp <nazwa-aplikacji>.war /opt/wildfly/standalone/deployments/`
- Sprawdź efekty:
  - Przeglądarka localhost:8080/<url-path>
  - curl -v localhost:8080/<url-path>
  - Konsola administracyjna Wildfly
    - *włączamy zbieranie statystyk*

Zadanie 4
Hello World Servlet

- Małe poprawki:
  - Zmieniamy nazwę artefaktu - ustawiamy ładniejszy context-root:

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

Zadanie 4
Serwujemy losowe liczby

- Zamiast “Hello World” wyświetlaj losową liczbę - użyj klasy NumbersGenerator

- Sprawdź efekty:
  - Przeglądarka
  - curl
  - Konsola administracyjna Wildfly

Zadanie 5
Serwujemy losowe liczby w HTMLu

- Zamiast samej liczby zwróć z servletu html:

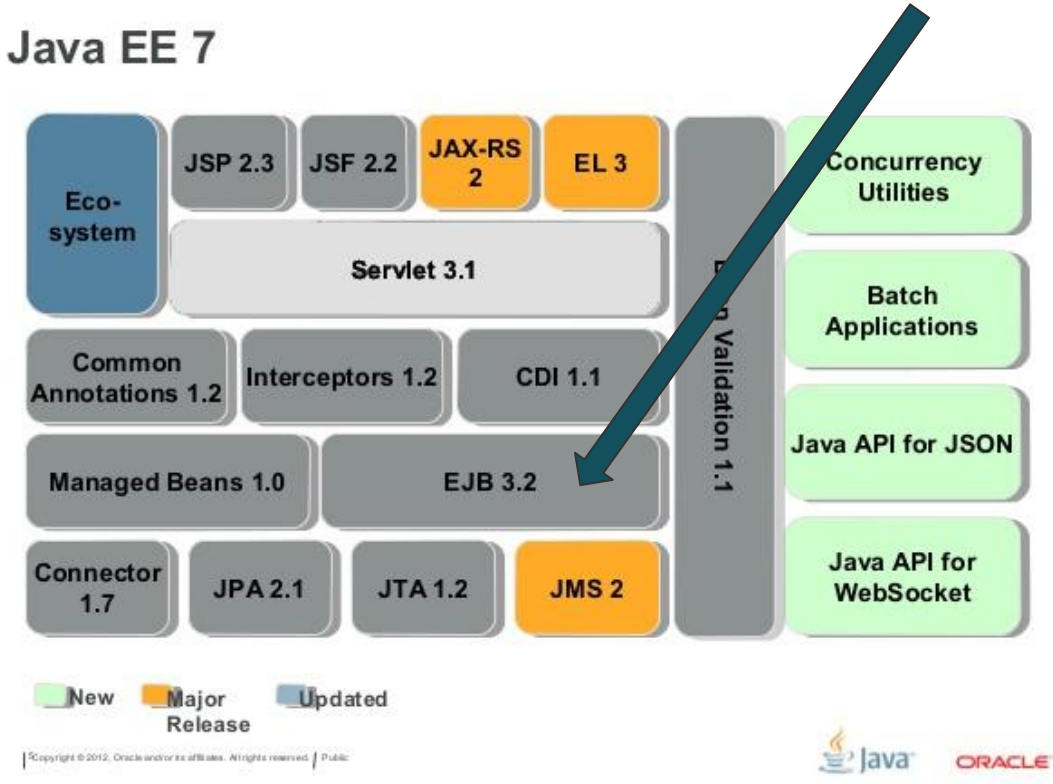
```
resp.setContentType("text/html;charset=UTF-8");

PrintWriter writer = resp.getWriter();

writer.println("<!DOCTYPE html>");
writer.println("<html>");
writer.println("<body text=\"green\">");
writer.println("Random number: "+numbers.getRandomInt(100));
writer.println("</body>");
writer.println("</html>");
```

4.
EJB

JEE 7


Java EE 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
Validation 1.1
Concurrency Utilities
Batch Applications
Java API for JSON
Java API for WebSocket
New
Major Release
Updated
Java
ORACLE

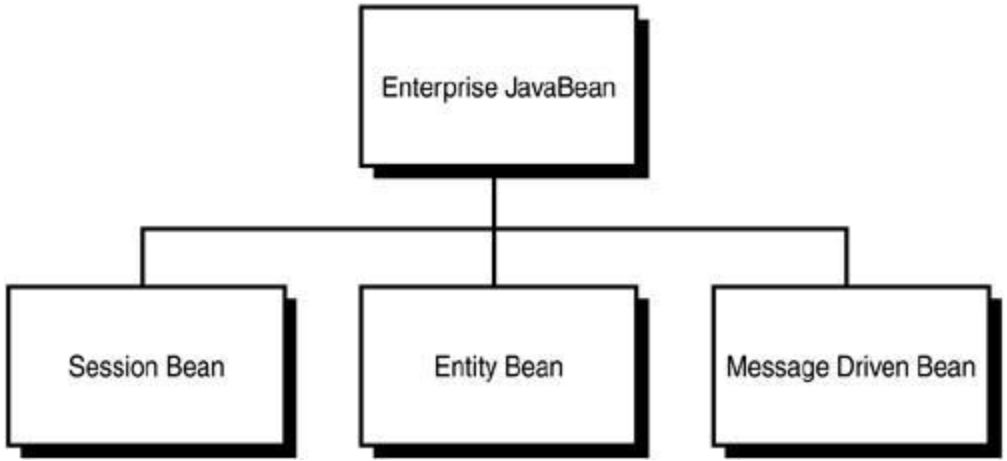
EJB
Enterprise Java Beans

- Co to jest *bean*?

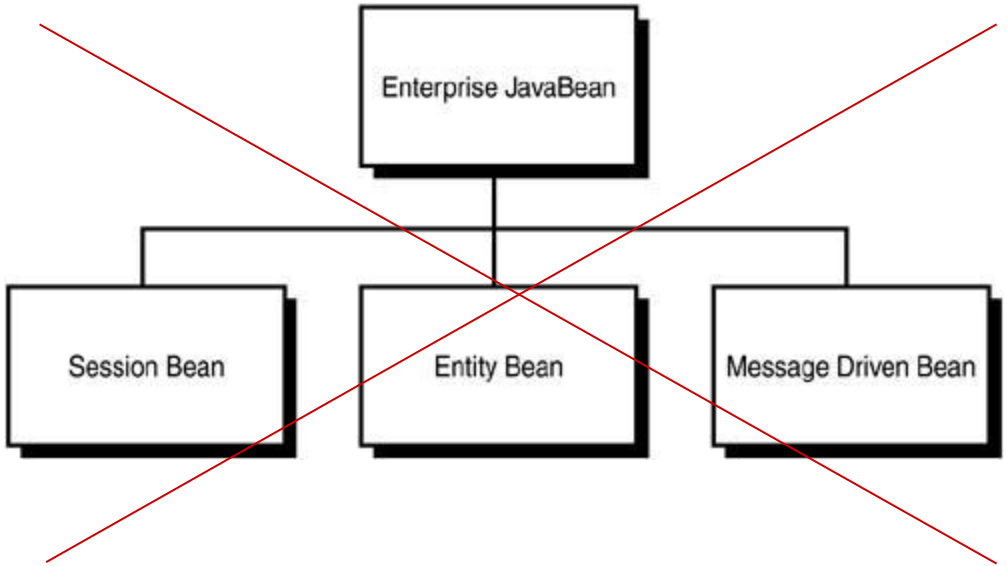
EJB
Enterprise Java Beans

- Jedna z najstarszych części specyfikacji JEE
- Warstwa biznesowa aplikacji - logika biznesowa
- Beanami EJB zarządza część serwera JEE zwana kontenerem EJB
- Kontener EJB realizuje za nas:
  - Lokalny i zdalny dostęp do beanów
  - Skalowalność
  - Transakcyjność
  - Odseparowanie warstwy logiki biznesowej od np. warstwy serwującej HTML

EJB
Enterprise Java Beans

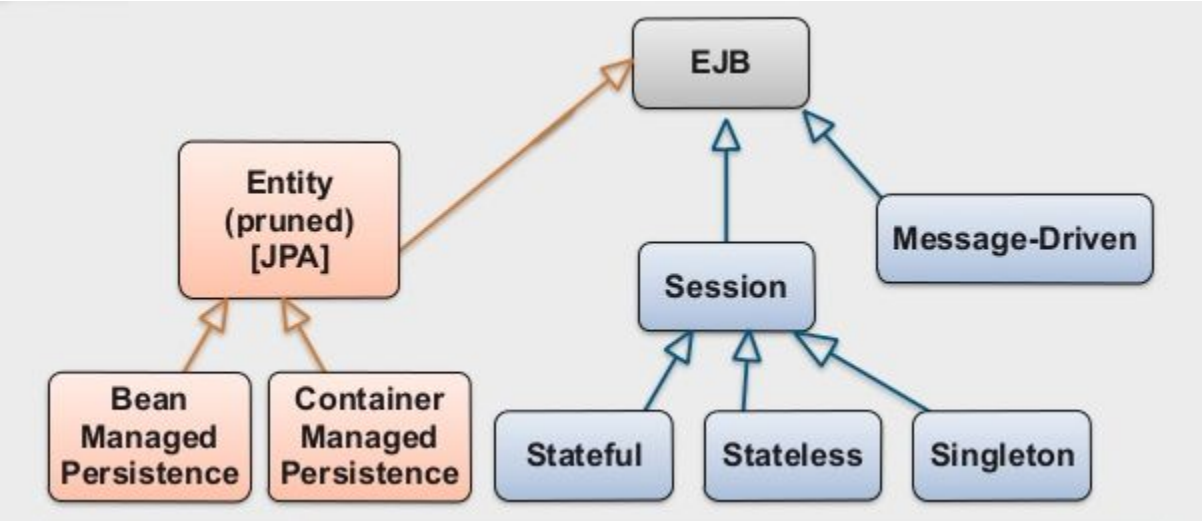


EJB
Enterprise Java Beans



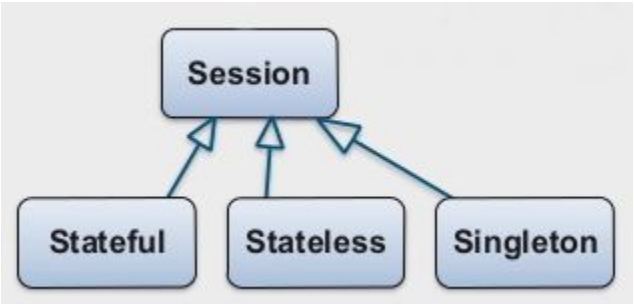
staroć

EJB
Enterprise Java Beans



EJB
Session Beans

- Mogą być wywołane przez lokalnych i zdalnych klientów
- 3 typy



EJB
Stateless Session Bean

- Implementuje interfejs oznaczony adnotacją *@Local* lub *@Remote*:

```
import javax.ejb.Remote;


@Remote
public interface AdderImplRemote {
int add(int a,int b);
}
```

EJB
Stateless Session Bean

- Jest oznaczony adnotacją *@Stateless*:

```
import javax.ejb.Stateless;


@Stateless
public class AdderImpl implements AdderImplRemote {
  public int add(int a,int b){
      return a+b;
  }
}
```

4.1
EJB - Local Interface

Zadanie 6
Stateless Session Bean z interfejsem lokalnym

- Zmień klasę *NumbersGenerator* w Stateless Session Bean z interfejsem lokalnym

Zadanie 6
Stateless Session Bean z interfejsem lokalnym

- Zmień klasę *NumbersGenerator* w Stateless Session Bean z interfejsem lokalnym:
  a. Dopisz interfejs lokalny *LocalNumbersGenerator*
  b. Zmień klasę *NumbersGenerator* w implementację interfejsu i dodaj odpowiednią adnotację

Zadanie 6
Stateless Session Bean z interfejsem lokalnym

- Zbuduj aplikację i uruchom na serwerze Wildfly
- Obserwuj logi podczas deploymentu
- Zajrzyj do konsoli administracyjnej

Zadanie 6
Stateless Session Bean z interfejsem lokalnym

- Zbuduj aplikację i uruchom na serwerze Wildfly
- Obserwuj logi podczas deploymentu
- Zajrzyj do konsoli administracyjnej
  - Zwróć uwagę na wielkość puli dla swojego beana

Zadanie 7
Zwiększamy pulę beanów stateless

- Przestaw wielkość puli dla stateless session beans na stałą wartość 20

Zadanie 7
Zwiększamy pulę beanów stateless

- Przestaw wielkość puli dla stateless session beans na stałą wartość 20:
  - Wejdź do katalogu /opt/wildfly/standalone/configuration/
  - Zrób backup pliku standalone-full.xml
  - Otwórz plik standalone-full.xml
  - Znajdź miejsce konfiguracji podsystemu *ejb*
  - Znajdź miejsce konfiguracji puli beanów stateless
  - Zmień konfigurację:

<strict-max-pool name="slsb-strict-max-pool" max-pool-size="20" instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>

Zadanie 7
Zwiększamy pulę beanów stateless

- Zmiana konfiguracji będzie widoczna po restarcie serwera
- Sprawdzamy w konsoli administracyjnej efekt naszych zmian

4.2
EJB - Dependency Injection

EJB
Dependency Injection

- Wzorzec architektury oprogramowania
- Polega na usuwaniu bezpośrednich zależności pomiędzy komponentami (np. klasami) na rzecz architektury typu plug-in

EJB
Dependency Injection

- Przekazywanie gotowych, utworzonych instancji obiektów do innych obiektów, które z nich korzystają
- Alternatywa do tworzenia obiektów przez obiekty z nich korzystające we własnym zakresie

EJB

Dependency Injection przez konstruktor

```
public class MyClass {

   private final static Calculator calculator;

   public MyClass() {
      this.calculator = new Calculator();
    }

    public  doSomething(int number) {
      calculator.abs(number)
   }
}
```

```
public class MyClass {

   private final static Calculator calculator;

   public MyClass(Calculator simpleCalculator) {
      this.calculator = simpleCalculator;
    }

    public  doSomething(int number) {
      calculator.abs(number)
   }
}
```

EJB
Dependency Injection

- Jeden ze sposobów realizacji wzorca projektowego *Inversion of Control*
- Zapewnia tzw. *loose coupling*
  - łatwe testowanie!

EJB

Dependency Injection przez kontener DI

- Kontenery w serwerze JEE dostarczają nam gotowe mechanizmy wstrzykiwania zależności
- Kontener, który potrafi zrobić *dependency injection* nazywany jest kontenerem DI lub kontenerem IoC

- Kontener EJB jest kontenerem DI

EJB
Kontener EJB jako kontener DI

- Kontener EJB daje nam adnotację @EJB:

```
public class MyClass {

   @EJB
   private final static CalculatorLocalInterface calculator;

   public MyClass() {
    }

    public  doSomething(int number) {
       calculator.abs(number)
   }
}
```

Zadanie 8

Wywołanie *stateless session beana z servletu*

- Dopisz wywołanie bean NumbersGenerator z servletu
  - skorzystaj z mechanizmu Dependency Injection

Zadanie 9

Wywołanie *stateless session beana* z servletu

- Włącz zbieranie statystyk dla podsystemu *ejb* w Wildfly (użyj konsoli administracyjnej)
- Sprawdź czy *bean* NumbersGenerator jest wywoływany kiedy korzystasz z aplikacji

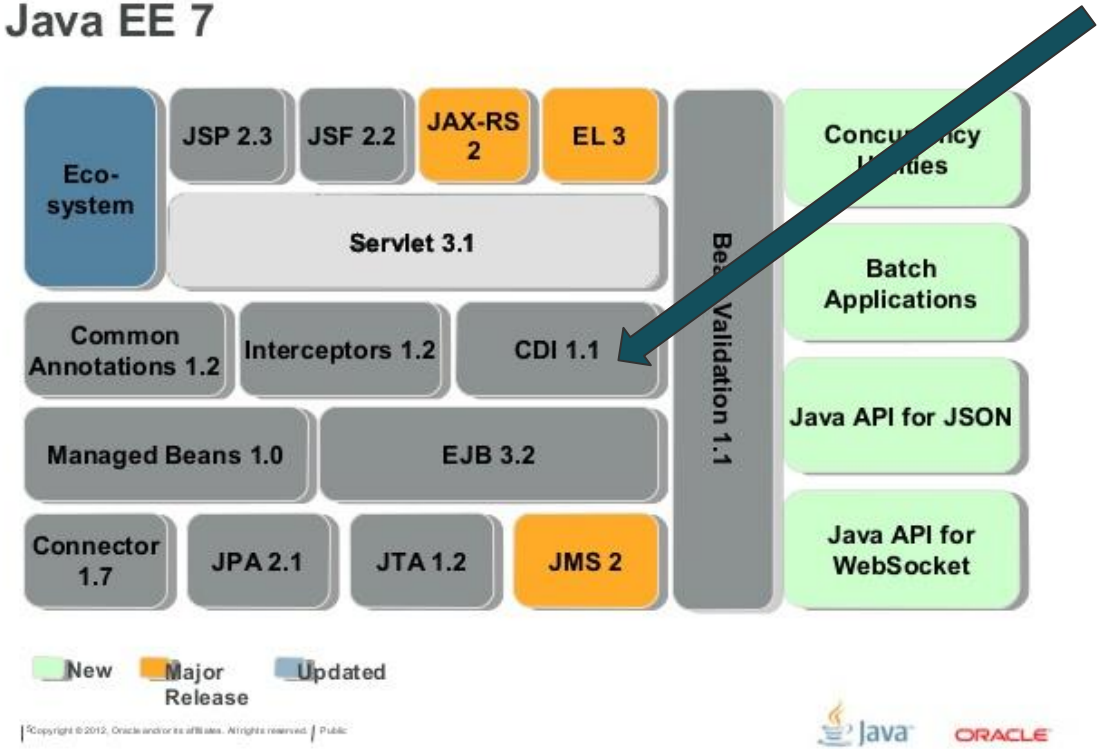
Zadanie 10
@Inject zamiast @EJB

- Zamień adnotację @EJB w servlecie na adnotację @Inject
- Sprawdź czy aplikacja nadal działa

CDI

- Adnotacja @EJB jest obsługiwana przez kontener EJB i pozwala na wstrzykiwanie tylko tych obiektów, którymi zarządza kontener EJB
- Adnotacja @Inject jest obsługiwana przez kontener CDI - *Context and Dependency Injection*

CDI


Java EE 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
New
Major Release
Updated
Java
ORACLE

4.3
EJB - Remote Interface

Zadanie 11
Stateless Session Bean z interfejsem zdalnym

- Dopisz interfejs zdalny dla klasy *NumbersGenerator*
- Uruchom na serwerze Wildfly i popatrz w logi
  - Co się zmieniło?

EJB
Stateless Session Bean z interfejsem zdalnym

- Interfejs zdalny sprawia, że można się odwołać do *beana* EJB zdalnie
  - Zdalnie czyli z innej maszyny wirtualnej (np. z komponentu uruchomionego na innym serwerze JEE lub z aplikacji standalone)

Zadanie 12.1
Wywołanie zdalne EJB z aplikacji standalone

- Stwórz nowy projekt i napisz aplikacje standalone, która wypisuje na konsolę “Hello World”
- Zbuduj, uruchom z linii poleceń i sprawdź czy wszystko działa

EJB
Stateless Session Bean z interfejsem zdalnym

- Żeby móc odwołać się do EJB z naszej aplikacji standalone potrzebujemy załączyć do niej bibliotekę zawierającą interfejs zdalny EJB

Zadanie 12.2
Wywołanie zdalne EJB z aplikacji standalone

- Stwórz bibliotekę dla klienta EJB zawierającą interfejs do *NumbersGenerator*

Zadanie 12.2
Wywołanie zdalne EJB z aplikacji standalone

- Stwórz bibliotekę dla klienta EJB zawierającą interfejs do *NumbersGenerator*
- Można do tego celu użyć Mavena i wtyczki maven-ejb-plugin

Zadanie 12.2
Wywołanie zdalne EJB z aplikacji standalone

- Dodaj konfigurację wtyczki do pom.xml

```
<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/isa/javaeebasics/randomgenerator/ejb/RemoteNumbersGenerator.class
            </clientInclude>
        </clientIncludes>
    </configuration>
</plugin>
```

Zadanie 12.2
Wywołanie zdalne EJB z aplikacji standalone

- Zbuduj projekt:

  ```
  mvn clean package
  ```

- Uruchom wtyczkę maven-ejb-plugin:

  ```
  mvn ejb:ejb
  ```

- Zajrzyj do katalogu /target
- Zainstaluj bibliotekę w lokalnym repozytorium mavena:

mvn install:install-file -Dfile=random-generator-client.jar -DgroupId=com.isa.jeebasics -DartifactId=random-generator-client -Dpackaging=jar -Dversion=1.0

Zadanie 12.3
Wywołanie zdalne EJB z aplikacji standalone

- Załącz bibliotekę *random-generator-client* jako zależność mavenową do aplikacji standalone

Zadanie 12.3
Wywołanie zdalne EJB z aplikacji standalone

- Załącz bibliotekę *random-generator-client* jako zależność mavenową do aplikacji standalone:

```
<dependency>
    <groupId>com.isa.jeebasics</groupId>
    <artifactId>random-generator-client</artifactId>
    <version>1.0</version>
</dependency>
```

Zadanie 12.3
Wywołanie zdalne EJB z aplikacji standalone

- Musimy zbudować aplikację tak, żeby zawierała załączoną bibliotekę wewnątrz JARa
  - Zbuduj aplikację standardowo:

    ```
    mvn clean package
    ```
  - Zajrzyj do środka JARa i sprawdź czy biblioteka tam jest:

    ```
    jar -tf <plik-z-aplikacja.jar>
    ```

Zadanie 12.3
Wywołanie zdalne EJB z aplikacji standalone

- Skonfiguruj mavena tak, żeby zbudował JARa zawierającego wszystkie biblioteki dołączone jako *dependencies*:
  - Usuń konfigurację wtyczki *maven-jar-plugin*
  - Użyj wtyczki *maven-assembly-plugin*

Zadanie 12.3

Wywołanie zdalne EJB z aplikacji standalone

```
<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.isa.javaeebasics.randomgenerator.client.App
                </mainClass>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Zadanie 12.3
Wywołanie zdalne EJB z aplikacji standalone

- Zbuduj projekt ponownie i sprawdź czy zawiera bibliotekę klienta

Zadanie 12.4
Wywołanie zdalne EJB z aplikacji standalone

- Skonfiguruj na serwerze Wildfly użytkownika i hasło dla zdalnego dostępu:

```
cd /opt/wildfly/bin
./add-user.sh
```

4.4
EJB - Remote Interface - JNDI lookup

JNDI
Java Naming and Directory Interface

- JNDI to specyfikacja API javowego, które ma służyć do wyszukiwania obiektów po nazwach
- JNDI nie jest częścią specyfikacji JEE, ale jest wykorzystywany w JEE - stąd serwery JEE implementują również JNDI
- Żeby skorzystać z JNDI do wyszukania obiektów na serwerze Wildfly trzeba użyć implementacji dostarczonej przez Wildfly

JNDI
Java Naming and Directory Interface

- Żeby dostać się do obiektów najpierw należy stworzyć obiekt klasy *InitialContext*
- *InitialContext* udostępnia metodę

  ```
  lookup(String nazwa)
  ```

  która umożliwia pobranie instancji obiektu
- Tworząc *InitialContext* podajemy jakiej konkretnej implementacji chcemy użyć i ustawiamy parametry właściwe dla tej implementacji

Zadanie 12.5
Wywołanie zdalne EJB z aplikacji standalone

- Załącz do aplikacji standalone'owego klienta EJB bibliotekę do łączenia się z serwerem Wildfly:

Zadanie 12.5
Wywołanie zdalne EJB z aplikacji standalone

- Załącz do aplikacji standalone'owego klienta EJB bibliotekę do łączenia się z serwerem Wildfly:

```
<dependency>
    <groupId>org.wildfly</groupId>
    <artifactId>wildfly-client-all</artifactId>
    <version>10.1.0.Final</version>
</dependency>
```

Zadanie 12.6
Wywołanie zdalne EJB z aplikacji standalone

- Stwórz *InitialContext* właściwy dla Wildfly 10 i skonfiguruj go tak, żeby mógł się połączyć do Twojego serwera Wildfly:

Zadanie 12.6
Wywołanie zdalne EJB z aplikacji standalone

- Stwórz *InitialContext* właściwy dla Wildfly 10 i skonfiguruj go tak, żeby mógł się połączyć do Twojego 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, "<hasło>");
Context context = new InitialContext(properties);
```

Zadanie 12.6
Wywołanie zdalne EJB z aplikacji standalone

- Wyszukaj *beana* NumbersGenerator po interfejsie zdalnym:

```
RemoteNumbersGenerator generator =
        (RemoteNumbersGenerator)
context.lookup("random-generator/NumbersGenerator!com.isa.javaeebasics.randomgenerator.ejb.RemoteNumbersGenerator");
```

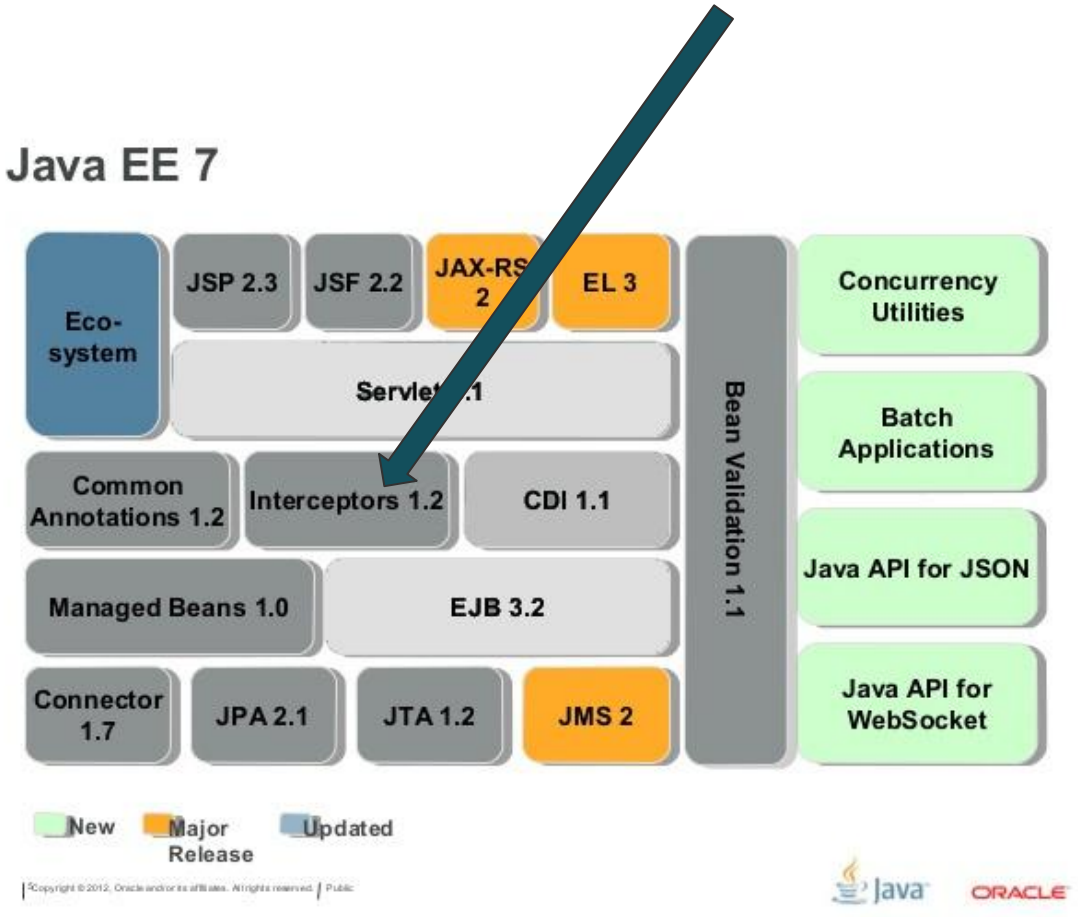
- Użyj go do wyświetlenia losowej liczby na konsolę

Zadanie 12.6
Wywołanie zdalne EJB z aplikacji standalone

- Uruchom aplikację i sprawdź czy działa
- Zajrzyj do konsoli administracyjnej Wildfly i zobacz czy widać, że *bean* jest wywoływany

5.
Interceptors

JEE 7


Java EE 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
New
Major Release
Updated
Copyright © 2012, Oracle and/or its affiliates. All rights reserved. | Public
Java
ORACLE

Interceptors
AOP

- AOP - Aspect Oriented Programming: sposób pisania programów komputerowych polegający na fizycznym rozdzieleniu różnych aspektów funkcjonalnych programu, które nie są ze sobą bezpośrednio związane
- Interceptory są jednym ze sposobów realizacji AOP

Każde realizowane zagadnienie pociąga za sobą w praktyce potrzebę realizacji zagadnień pobocznych. Na przykład program przelewający pieniądze na kontach bankowych realizuje nie tylko swój główny cel (tj. pomniejszenie zawartości jednego konta i powiększenie zawartości drugiego), ale równocześnie z nim również zagadnienia logowania, bezpieczeństwa, spójności transakcyjnej, autoryzacji, synchronizacji wielowątkowej i wiele innych. Zagadnienia te są w dużym stopniu rozłączne pomiędzy sobą pod względem funkcjonalnym. Aby je zrealizować, programista musi poprzeplatać ich implementacje (tzw. warkocz), co czyni kod mniej czytelnym, bardziej podatnym na błędy, trudniejszym w modyfikacji.

AOP zapobiega tym negatywnym skutkom oddzielając fizycznie kod każdego zagadnienia poprzez umieszczenie ich w oddzielnych aspektach i logiczne zdefiniowanie punktów interakcji pomiędzy nimi.

Interceptors

- Interceptor to w pewnym sensie obserwator, do którego zostanie przekazane sterowanie np. w momencie wywołania obserwowanej metody

Interceptors

```
public class MyClass {

    private final static Calculator calculator;

    @Interceptors(SomeInterceptor)
    public  doSomething(int number) {
        calculator.abs(number)
    }
}
```

```
public class SomeInterceptor {

    @AroundInvoke
    public  Object intercept(InvocationContext context) throws
Exception {
        logger.info("Method was invoked.");
        context.proceed();
    }
}
```

Interceptors

```
public class MyClass {

    private final static Calculator calculator;

    @Interceptors(SomeInterceptor.class)
    public  doSomething(int number) {
        calculator.abs(number)
    }
}
```

metoda obserwowana

klasa interceptora

```
public class SomeInterceptor {

    @AroundInvoke
    public Object intercept(InvocationContext context) throws
Exception {
        logger.info("Method was invoked.");
        context.proceed();
    }
}
```

metoda interceptora

Zadanie 13
Interceptor

- Zaimplementuj interceptor *TimeLoggingInterceptor* który przechwyci wywołanie metody *getRandomInt* beana *NumbersGenerator*
- Interceptor powinien zalogować czas wykonania metody
  - Użyj biblioteki slf4j do logowania

Zadanie 14
Interceptor

- Zaimplementuj interceptor *ParametersLoggingInterceptor* który przechwyci wywołanie metody *getRandomInt* beana *NumbersGenerator* i zaloguje parametry, z jakimi została wywołana metoda

Zadanie 15
Interceptor

- Zmień kolejność wywołania interceptorów tak, aby najpierw logowane były parametry wywołania

Thanks!!

Any questions?