SOA – laboratorium nr 4

Temat: Tworzenie EJB oraz aplikacji klienckich.

EJB 3.2 — nowe funkcjonalności

Według specyfikacji, **Enterprise JavaBeans (EJB)** to komponenty, których podstawowym zadaniem w aplikacjach **Java Enterprise Edition (JEE)** jest implementacja logiki biznesowej i dostępu do danych.

W zasadzie istnieją trzy rodzaje komponentów EJB:

- **bezstanowe ziarna sesyjne** (SLSB — *Stateless Session Beans*) — obiekty, których instancje nie zawierają żadnych informacji o stanie konwersacji, więc gdy nie obsługują aktualnie konkretnego klienta, w zasadzie są sobie równoważne;
- **stanowe ziarna sesyjne** (SFSB — *Stateful Session Beans*) — obiekty obsługujące usługi konwersacyjne dotyczące silnie powiązanych klientów; stanowe ziarno sesyjne wykonuje zadania dla konkretnego klienta i przechowuje stan przez cały czas trwania sesji z klientem; po zakończeniu sesji stan nie jest dłużej przechowywany;
- **ziarna sterowane komunikatami** (MDB — *Message-Driven Beans*) — rodzaj komponentu EJB mogący asynchronicznie przetwarzać komunikaty przesyłane przez dowolnego producenta JMS

Poza standardowymi komponentami EJB, serwer aplikacji obsługuje również nowe odmiany EJB 3.2 wprowadzone wraz z Javą EE 6.

- **Singletonowy komponent EJB** — przypomina bezstanowe ziarno sesyjne, ale do obsługi żądań klientów wykorzystywana jest tylko jedna instancja, co gwarantuje użycie tego samego obiektu we wszystkich wywołaniach. Singletony mogą korzystać z bogatszego cyklu życia dla pewnego zbioru zdarzeń, a także ze ściślejszych zasad blokad, by prawidłowo obsłużyć współbieżny dostęp do instancji.
- **Bezinterfejsowy komponent EJB** — to nieco inne spojrzenie na standardowe ziarno sesyjne, bo od lokalnych klientów nie wymaga się osobnego interfejsu, czyli wszystkie metody publiczne klasy ziarna są dostępne dla kodu wywołującego.
- **Asynchroniczne komponenty EJB** — umożliwiają przetwarzanie żądań klientów w sposób asynchroniczny (podobnie jak w przypadku MDB), ale udostępniają typowany interfejs i stosują nieco bardziej wyrafinowane podejście do obsługi żądań klientów, które dzieli się na dwa etapy:

Pierwsza komponent sesyjny w JavaEE.

Tworzenie komponentów EJB w JAVA EE nie jest zbyt skomplikowane. Sprowadza się w zasadzie do stworzenia zwykłych klas, które posiadając odpowiednie adnotacje.

Stwórzmy więc dwa komponenty: Bezstanowy – zwracający zawsze jakąś wartość np. niech to będzie wynik dodawania dwóch liczb oraz Singleton – będący licznikiem wykonanych obliczeń.

Nasza aplikacja składać się będzie z 3 niezależnych części:

- implementacji komponentów

- specyfikacji interfejsów

oraz implementacji klienta wykorzysującego nasze komponenty w postaci apliakacji webowej w postaci servleta lub pliku JSF. .

Tworzymy wiec 3 projekty: (Do ich tworzenia można użyć mavena lub stworzyć je samodzielnie)

ejb3-server-api	ejb3-server-impl	ejb3-server-war
Rodzaj deploy : jar	**Rodzaj deploy:** jar	**Rodzaj deploy:** war

Projekt **ejb3-server-api** zawiera tylko specyfikacje interfejsów: w zależności od planowanego użycia należy stworzyć interfejsy lokalne lub zdalne (lub takie i takie)

ITestAddBean :

```
package pl.agh.kis.soa.ejb3.server.api;

public interface ITestAddBean {

        int add(int a,int b);
}
```

- ILocalTestAddBean :

```
package pl.agh.kis.soa.ejb3.server.api;

public interface ILocalTestAddBean extends ITestAddBean {}
```

IRemoteTestAddBean :

```
package pl.agh.kis.soa.ejb3.server.api;

public interface IRemoteTestAddBean extends ITestAddBean {}
```

ITestBeanCounter :

```
package pl.agh.kis.soa.ejb3.server.api;

public interface ITestBeanCounter {

        void increment();
        long getNumber();
}
```

ILocalTestBeanCounter :

```
package pl.agh.kis.soa.ejb3.server.api;

public interface ILocalTestBeanCounter extends ITestBeanCounter {}
```

IRemoteTestBeanCounter :

```
package pl.agh.kis.soa.ejb3.server.api;

public interface IRemoteTestBeanCounter extends ITestBeanCounter{}
```

Projekt **ejb3-server-imp** zawiera implementacje poszczególnych interfejsów

```
package pl.agh.kis.soa.ejb3.server.impl;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

@Stateless
@LocalBean
public class TestAddBean implements ILocalTestAddBean {

    /**
     * Default constructor.
     */
    public TestAddBean () {
        // TODO Auto-generated constructor stub
    }

    public int add(int a,int b){
     int r=a+b;
     return r;
    }

}
```

```
package pl.agh.kis.soa.ejb3.server.impl;

import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.Singleton;

import pl.agh.kis.soa.ejb3.server.api.ILocalTestBeanCounter;
import pl.agh.kis.soa.ejb3.server.api.IlTestBeanCounter;
import pl.agh.kis.soa.ejb3.server.api.IRemoteTestBeanCounter;
```

```
@Singleton
@Remote(IRemoteTestBeanCounter.class)
@Local(ILocalTestBeanCounter.class)
public class TestBeanCounter implements ITestBeanCounter{

    long counterNumber = 0;

@Override
public void increment() {
counterNumber ++;
}

@Override
public long getNumber() {
return counterNumber;
}

}
```

Zamiast deklaracji lokalnego interfejsu możliwe jest zastosowanie @LocalBean :

```
@Singleton
@Remote(IRemoteTestBeanCounter.class)
@LocalBean
public class TestBeanCounter implements ITestBeanCounter{
        //...
}
```

Dzieki temu w aplikacji klienckiej możliwe jest bezpośrednie wstrzyknięcie komponentu poprzez uzycie:

```
@EJB
TestBeanCounter
```

Innym sposobem dostępu do komponenty z poziomu aplikacji klienckiej jest użycie JNDI i jej metody lookup.

Poniższa tabela pozwoli zorientować się w znaczeniu poszczególnych elementów.

Element	**Opis**
app-name	To nazwa aplikacji typu enterprise (bez elementu .ear), jeśli komponent EJB znajduje się w pakiecie EAR.
module-name	To nazwa modułu (bez elementu .jar lub .war), w którym znajduje się komponent EJB
distinct-name	Można opcjonalnie ustawić nazwę wyróżniającą dla każdej jednostki wdrożenia. bean-name To nazwa klasy ziarna.
fully-qualified-classname-of-the-remote-interface	To w pełni kwalifikowana nazwa klasy interfejsu zdalnego

Przykładowy kod pozwalający na wyszukiwanie komponentów stanowych wyglądałby mniej więcej tak:

```
import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class LookerUp {

private Properties prop = new Properties();
private String jndiPrefix;

public LookerUp(){
prop.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
}
public Object findLocalSessionBean(String moduleName, String beanName, String interfaceFullQualified
Name) throws NamingException{

final Context context = new InitialContext(prop);
Object object = context.lookup("java:global/"+moduleName+"/"+beanName+"!"+interfaceFullQualifiedNa
me);
context.close();

return object;
}

public Object findSessionBean(String jndiName) throws NamingException{

final Context context = new InitialContext(prop);
Object object = context.lookup(jndiName);
context.close();

return object;
}

}
```

Wywołanie w kodzie klienta:

```
//--- EJB Lookup w tym samym  WAR
String moduleName = "ejb3-server-client-war"; // WAR name
String beanName = "TestBean";
String interfaceQualifiedName = ILocalTestBean.class.getName();
LookerUp wildf9Lookerup = new LookerUp();
proxy = (ILocalTestBean) wildf9Lookerup.findLocalSessionBean(moduleName,beanName,interfaceQualif
iedName);
```

Samodzielnie dokończ projekt implementując aplikacje webową, która korzystać będzie z obu komponentów.

Zadanie zaliczeniowe

1. Napisać aplikacje do zakupu biletów do teatru w oparciu o różnego rodzaju komponenty EJB.

Singletonowy komponent EJB ma zawierać metody obsługujące zarządzanie miejscami w teatrze. Dodajmy do projektu kilka ziaren sesyjnych związanych z logiką biznesową, takich jak bezstanowe ziarno sesyjne odpowiedzialne za informacje o dostępności poszczególnych miejsc w teatrze i stanowe ziarno sesyjne działające jako pośrednik systemu płatności – pozwalające na zakup biletu na określone miejsce. Zakup wiąze się z zmniejszeniem stanu konta poszczególnego użytkownika. .

Ziarno singletonowe udostępnia trzy metody publiczne. Metoda getSeatList zwraca listę obiektów Seat, które zostaną wykorzystane do wskazania użytkownikowi, czy podane miejsce zostało zarezerwowane.

Metoda getSeatPrice to metoda pomocnicza, która zwraca cenę za miejsce jako typ int, co umożliwia szybkie sprawdzenie, czy użytkownika stać na zakup wskazanego miejsca.

Ostatnia z metod, buyTicket, odpowiada za zakup biletu i oznaczenie miejsca jako zarezerwowanego.

Oprócz tego Singleton ma stworzyć liste miejsc z przypisanymi im cenami w momencie stworzenia komponentu.

Ziarno nad metodami dotyczącymi obsługi obiektów Seat powinno zawiera adnotację **@Lock**. **Służy ona do sterowania współbieżnością singletonu.** Współbieżny dostęp do singletonowego EJB jest domyślnie kontrolowany przez kontener.

Aby kontrolować zawartość portfela klienta, potrzebny będzie komponent przechowujący dane sesji z klientem. Głównym celem klasy sesyjnej jest wywołanie metody buyTicket singletonu po przeprowadzeniu kilku prostych testów związanych z logiką biznesową. Jeśli w trakcie sprawdzeń pojawi się sytuacja niedozwolona, aplikacja zgłosi wyjątek. Dotyczy to między innymi sytuacji, w których miejsce zostało już zarezerwowane lub gdy klient nie posiada wystarczających środków na zakup biletu