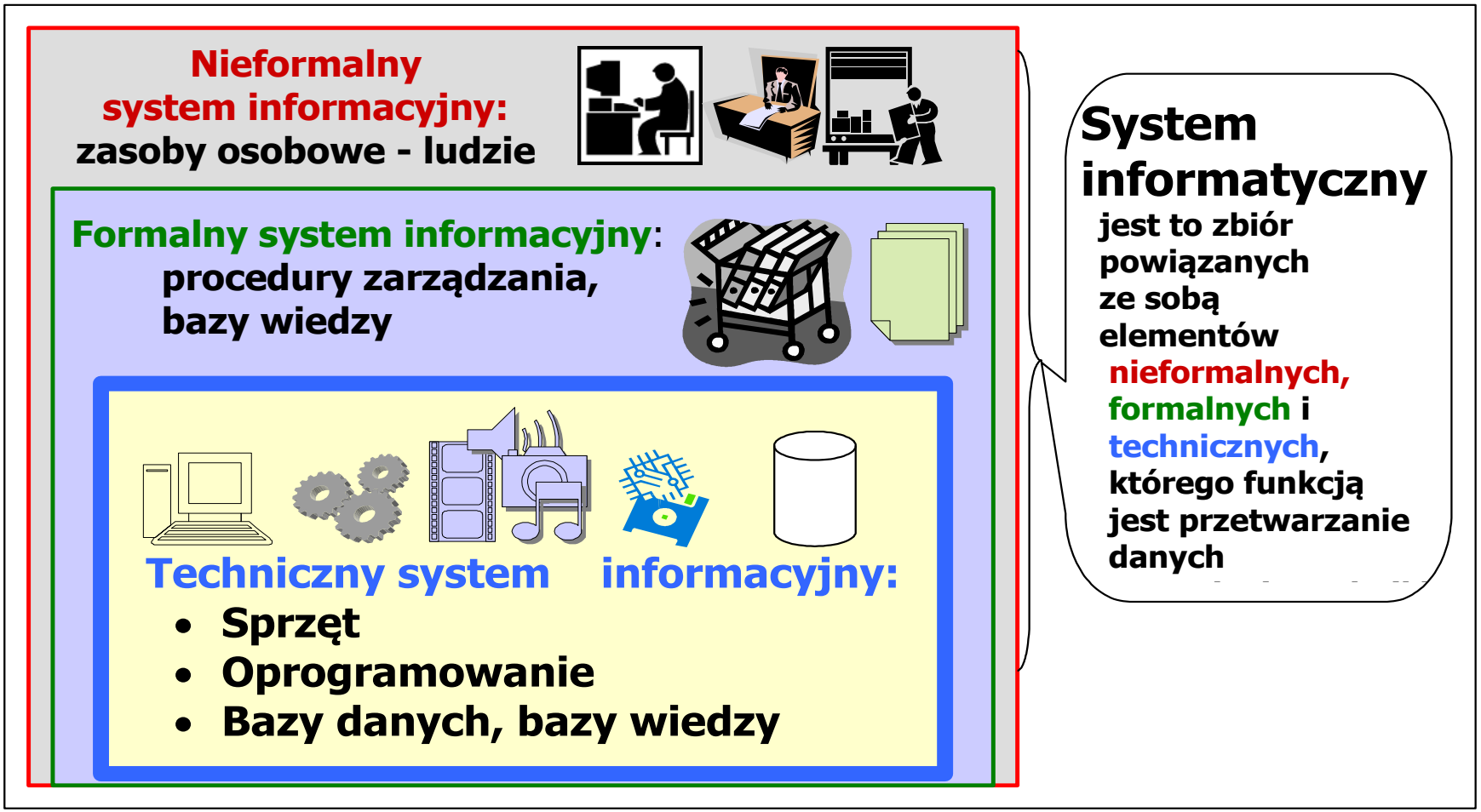
Podstawowe informacje o technologii Java EE 7

na podstawie
https://docs.oracle.com/javaee/7/JEETT.pdf

Programowanie komponentowe 2

I. Wielowarstwowa architektura systemu informatycznego

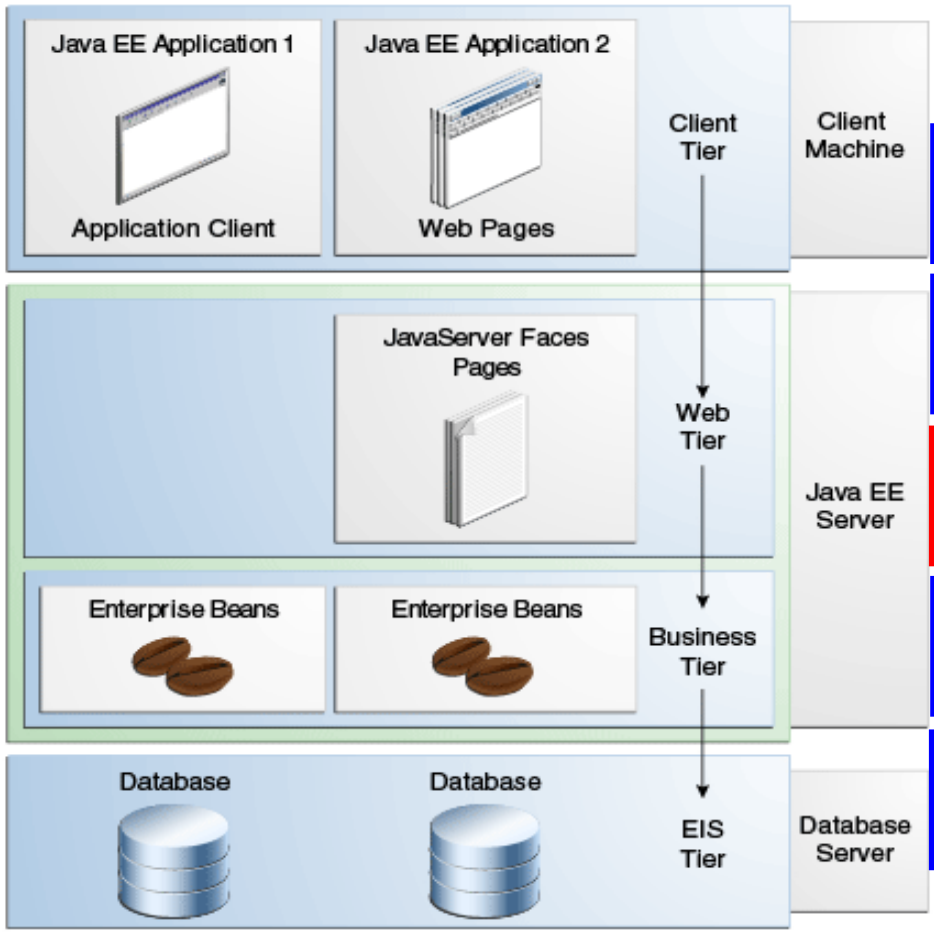
Definicja systemu informatycznego



Techniczny system informacyjny

- zorganizowany zespół środków technicznych (komputerów, oprogramowania, urządzeń teletransmisyjnych itp.)
- służący do gromadzenia, przetwarzania i przesyłania informacji

Java EE 7 ze strony https://docs.oracle.com/javaee/7/tutorial
Pięciowarstwowy model logicznego rozdzielania zadań



Warstwa klienta
Klienci aplikacji, aplety, aplikacje i inne elementy z graficznym interfejsem użytkownika

Interakcja z użytkownikiem, urządzenia i prezentacja interfejsu użytkownika

Warstwa prezentacji
Strony JSP, serwlety i inne elementy interfejsu użytkownika

Logowanie, zarządzanie sesją, tworzenie zawartości, formatowania i dostarczanie

Warstwa biznesowa
Komponenty EJB i inne obiekty biznesowe

Logika biznesowa, transakcje, dane i usługi

Warstwa integracji
JMS, JDBC, konektory i połączenia z systemami zewnetrznymi

Adaptery zasobów, systemy zewnętrzne, mechanizmy zasobów, przepływ sterowania

Warstwa zasobów
Bazy danych, systemy zewnętrzne i pozostałe zasoby

Zasoby, dane i usługi zewnętrzne

II. Komponenty typu EJB

Komponent – element do budowy warstw

- skompilowany moduł programowy,
- funkcjonalność dostarczana za pomocą interfejsu biznesowego,
- zdolny do współdziałania z innymi komponentami oraz innymi częściami systemu informatycznego.

1. Enterprise Bean – implementacja technologii Enterprise JavaBeans (EJB)

1) **Komponent napisany** w języku Java **do hermetyzacji logiki biznesowej** po stronie serwera.

2) **Logika biznesowa** spełnia podstawowe funkcje aplikacji i jest używana przez warstwę klienta aplikacji za pomocą komponentu obsługującego warstwę klienta lub za pośrednictwem komponentów warstwy internetowej.

np Managed_produkt obsługujący warstwę internetową aplikacji używa metod komponentu typu Fasada_warstwy_biznesowej:

- **public void utworz_produkt(String[] dane, Date data**),
- public dane_produktu(),
- public ArrayList<ArrayList<String>> Items()

1.1. Zalety użycia komponentów typu Enterprise Beans (EJB)

Komponenty EJB ułatwiają tworzenie dużych rozproszonych aplikacji:

1) **kontener komponentu EJB obsługuje usługi na poziomie systemowym**, np: tranzakcje, autoryzacja. Programista może **skoncentrować się na rozwiązaniu problemów logiki biznesowej**
2) **Komponenty EJB w przeciwieństwie do warstwy klienta (GUI desktopowe lub strona www)** zawierają kod logiki biznesowej. Powoduje to, że programiści warstwy klienta skupiają się funkcjonalności formularzy GUI internetowego lub desktopowego, możliwych do uruchomienia na małych urządzeniach
3) **Komponety EJB są komponentami przenośnymi** - można budować nowe aplikacje z wykorzystaniem istniejących komponentów EJB na kompatybilnym serwerze

1.2. Kiedy stosować komponenty EJB

Komponenty EJB są używane wtedy, gdy tworzona aplikacja musi spełniać następujące wymagania:

1) **Konieczność zapewnienia skalowalności aplikacji.** Przy wzrastającej liczbie użytkowników aplikacji, można komponenty EJB umieścić na wielu maszymach. Dla warstwy klienta jest to niewidoczne - różne serwery oraz ich lokalizacja.
2) **Transakcje muszą zapewnić integralność danych**. Komponenty EJB wspierają obsługę tranzakcji przy równoczesnym dostępie wielu instancji warstwy klienta
3) **Aplikacja może obsługiwać wiele różnych typów instancji warstwy klienta**. Dodając jedynie kilka linii kodu, wiele instancji warstwy klienta mogą w łatwy sposób lokalizować zdalne komponenty EJB.

1.3. Typy komponentów EJB

Typ komponentu EJB	Cel
Session	Wykonuje zadania dla warstwy klienta aplikacji. Opcjonalnie, może implementować usługi internetowe
Message-driven	Działa jako detektor określonego typu wiadomości, takich jak API Java Message Service

2. Czym jest komponent typu Session Bean

1) Komponent typu **Session Bean hermetyzuje logikę biznesową** i jest umieszczony na serwerze w warstwie biznesowej aplikacji, która może być używana programowo przez warstwę klienta: **lokalnie, zdalnie lub jako usługa internetowa**. W celu dostępu do aplikacji po stronie serwera, warstwa klienta wywołuje metody komponentu typu Session Bean.

2) Warstwa klienta, umieszczona na innym komputerze **jest niezależna** od złożoności działania usług biznesowych po stronie serwera.

3) Komponent typu Session Bean **nie jest utrwalany** w bazie danych

2.1. Rodzaje komponentów typu Session Bean - stateful, stateless i singleton.

2.1.1. Stateful (stanowy) komponent typu Session Bean

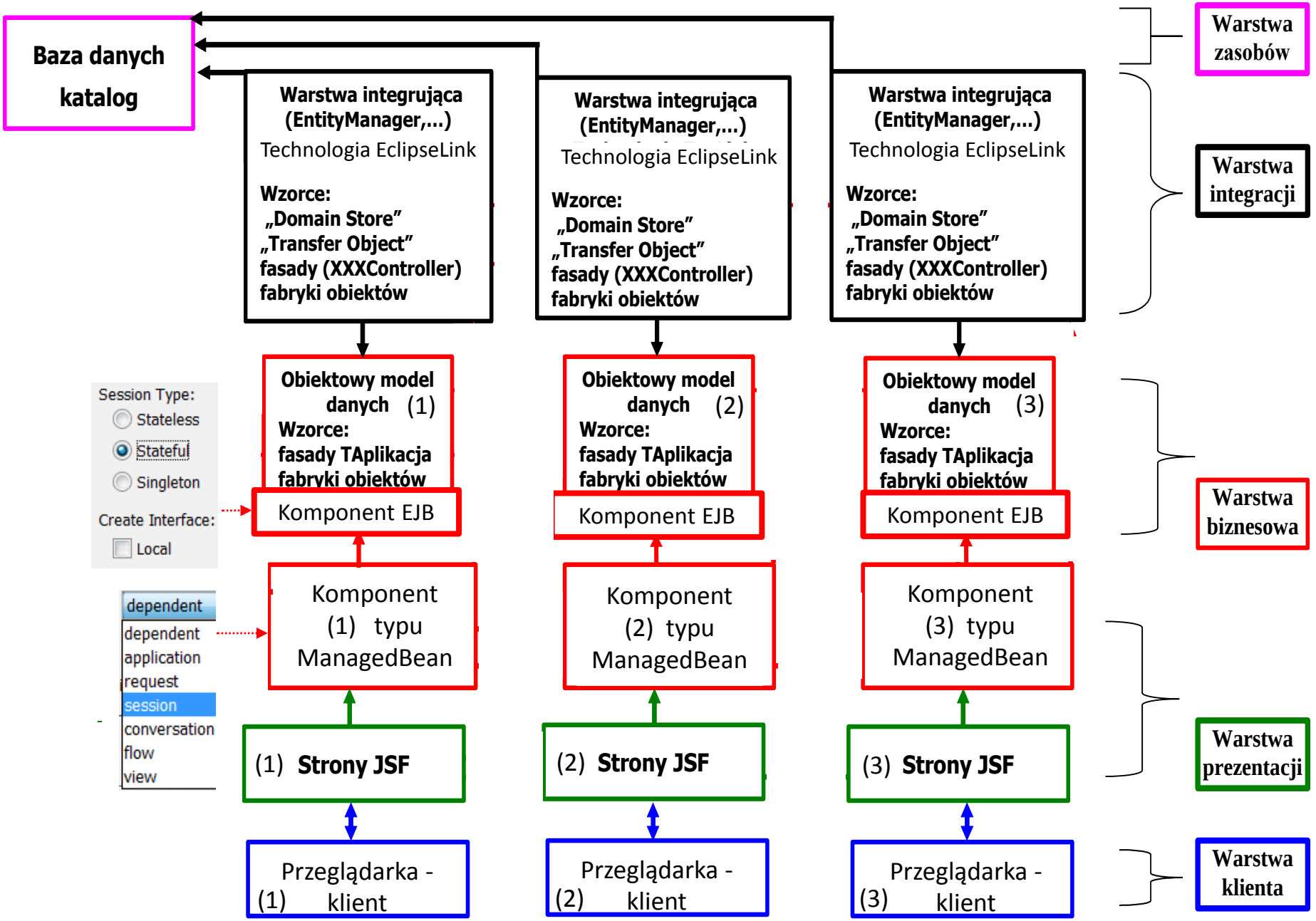
Stan obiektu składa się z wartości jego atrybutów. W stanowej sesji komponentu wartości atrybutów komponentu przedstawiają unikatowy stan komponentu warstwy klienta.

Ponieważ klient komunikuje się ("rozmawia") z komponentem, stan ten nazywany jest często **stanem konwersacji.**

Sesja komponentu nie jest współdzielona - może obsługiwać tylko jedną instancję warstwy klienta. Gdy instancja warstwy klienta zamyka sesję, komponent kończy swoje działanie.

(1) Przykłady architektury wielowarstwowej aplikacji internetowej typu EE opartej na komponentach stanowych (stateful) typu Session Bean

Architektura aplikacji pięciowarstwowej – Java EE 7.0 JavaServer Faces

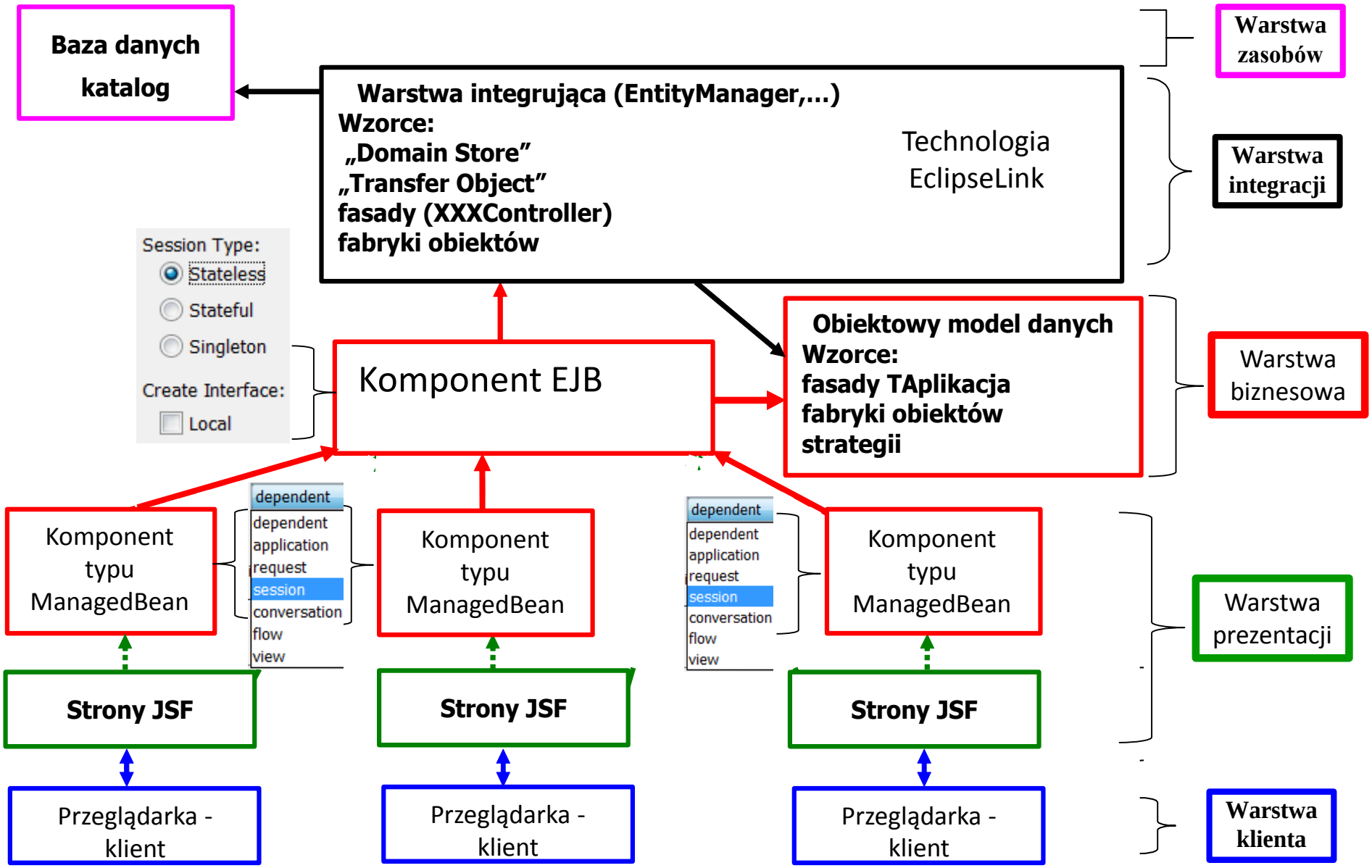


2.1.2 Stateless (bezstanowe) komponenty typu Session Bean

1) **Bezstanowa sesja komponentu nie utrzymuje stanu konwersacji z instancją warstwy klienta.** Gdy warstwa klienta wywołuje metody bezstanowego komponentu, wartości atrybutów komponentu mogą zawierać stan specyficzny dla danej instancji warstwy klienta tylko w czasie trwania wywołania. Gdy wywołana metoda jest zakończona, stan wywołującej warstwy klienta nie jest zachowany. Instancje warstwy klienta mogą jednak zmienić stan atrybutów bezstanowego komponentu i stan ten utrzymuje się do następnego wywołania metody tego komponentu.
2) **Komponenty bezstanowe mogą obsługiwać wielu klientów, umożliwiają dużą skalowalność aplikacji**. Mogą one obsłużyć większą ilość instancji klienta niż komponenty stanowe (stateful).
3) **Bezstanowe komponenty mogą implementować usługi internetowe, w przeciwieństwie do stanowych komponentów**.

(2) Przykłady architektury wielowarstwowej aplikacji internetowej typu EE opartej na komponentach bezstanowych (stateless) typu Session Bean

Architektura aplikacji pięciowarstwowej -Java EE 7.0 JavaServer Faces



2.1.3. Singleton - komponent typu Session Bean

1) **Komponent typu Singleton jest tworzony raz w aplikacji i istnieje do końca czasu działania aplikacji.**
2) Singleton jest stosowany wtedy, gdy jako jedyny jest używany współbieżnie przez wiele instancji warstwy klienta.
3) Singleton oferuje podobną funkcjonalność do komponentów bezstanowych, ale różnią się od nich tym, że istnieje tylko jedna instancja komponentu, w przeciwieństwie do puli komponentów bezstanowych, z których każdy może odpowiedzieć na żądanie klienta.
4) Podobnie jako komponenty bezstanowe, Singleton może **implementować usługę internetową.**
5) Singleton utrzymuje swój stan pomiędzy wywołaniami klienta, ale nie jest wymagane utrzymanie stanu podczas awarii serwera lub przestojów.
6) Aplikacje korzystające z komponentów typu Singleton używają go do **wykonania czynności inicjujących aplikacji**. Podobnie, komponent typu Singleton może **wykonywać czynności „czyszczące" podczas zamykania aplikacji.**

2.2. Kiedy należy używać komponenty typu Session Bean

Kontekst użycia stanowych (statefull) komponentów Session Bean:

1) Stan komponentu reprezentuje interakcje z warstwą klienta
2) Komponent musi utrzymać informację o warstwie klienta podczas wykonywania wywołanej metody przez warstwe klienta.
3) Komponent pośredniczy między klientem i innymi komponentami reprezentując uproszczony widok tej warstwy klienta.
4) Komponent może zarządzać przepływem pracy innych komponentów EJB.

2.2. Kiedy należy używać komponenty typu Session Bean (cd)

Kontekst użycia bezstanowych (stateless) komponentów Session Bean - w celu poprawy wydajności, jeśli:

1) Komponenty nie muszą przechowywać w swoich atrybutach danych instancji warstwy klienta.
2) W pojedynczym wywołaniu komponent wykonuje ogólne zadanie dla wszystkich klientów np w celu wyslania email z potwierdzeniem złożenia zamówienia on-line.
3) Komponent musi implementować usługę internetową.

2.2. Kiedy należy używać komponenty typu Session Bean (cd)

Kontekst użycia komponentów Singleton typu Session Bean stosuje się, jeśli:

1) Stan tego komponentu musi reprezentować stan aplikacji.
2) Pojedynczy komponent musi być dostępny dla wielu wątków.
3) Aplikacja potrzebuje komponentów typu Singleton w celu wykonywania zadań dotyczących uruchamiania i zamykania aplikacji.
4) Komponent realizuje usługę internetową.

2.3 Czym jest komponent typu Message-Driven Bean

1) Jest to komponent, który umożliwia aplikacjom Java EE **przetwarzać wiadomości asynchronicznie.**
2) Działa **jako słuchacz wiadomości** JMS (Java Message Service)**, podobnie jak detektor zdarzeń, ale odbiera komunikaty JMS zamiast zdarzeń. Komunikaty mogą być wysyłane przez każdy komponent Java EE** (klient aplikacji komponentu EE lub komponentu internetowego) lub za pomocą aplikacji JMS lub systemu, który nie korzysta z technologii Java EE.
3) **Komponent typu Message-Driven Bean przetwarza wiadomości JMS lub inne rodzaje komunikatów.**

2.3.1. Czym różnią się komponenty typu Message-Driven Beans od komponentów typu Session Beans?

Główna różnica między komponentami sterowanymi komunikatami i komponentami typu Session jest to, że warstwa klienta nie ma dostępu do metod komponentów typu Message-Driven.

Komponenty typu Message-Driven w kilku aspektach przypomina zachowanie komponentów bezstanowych typu Session Bean:

1) Komponenty typu Message-Driven nie przechowują danych konkretnej warstwy klienta.
2) Wszystkie instancje komponentów typu Message-Driven są równoważne, dzięki czemu kontener EJB może przypisać wiadomość do dowolnego komponentu typu Message-Driven. Kontener może połączyć te instancje, aby umożliwić równoległe przetwarzanie strumienia wiadomości.
3) Pojedynczy komponent typu Message-Driven może przetwarzać wiadomości wielu nadawców wiadomości.

2.3.1. Czym różnią się komponenty typu Message-Driven Beans od komponentów typu Session Beans? (cd)

Komponenty typu Message-Driven mogą zawierać stany reprezentujące :

- połączenia JMS API,
- otwarte połączenia z bazą danych
- odwołania do komponentów typu EJB.

Komponenty klienckie nie mogą zlokalizować komponentów typu Message-Driven i wywoływać metod tych komponentów.

Zamiast tego, klient uzyskuje dostęp do komponentu typu Message-Driven za pośrednictwem komunikatów JMS poprzez wysłanie wiadomości do komponentu Message-Driven typu MessageListener. Przypisanie odbiorcy wiadomości następuje podczas tworzenia bajtkodu dla serwera typu GlassFish.

2.3.1. Czym różnią się komponenty typu Message-Driven Beans od komponentów typu Session Beans? (cd)

Komponenty typu Message-Driven mają następujące cechy;

1) działają po otrzymaniu pojedynczej wiadomości
2) działają asynchronicznie
3) mają relatywnie krótki czas życia
4) nie reprezentują bezpośrednio udostępnionych dane w bazie danych, ale mają dostęp i aktualizują te dane
5) mogą świadomie dokonywać transakcji
6) są bezstanowe.

2.3.1. Czym różnią się komponenty typu Message-Driven Beans od komponentów typu Session Beans? (cd)

1) Gdy przychodzi wiadomość, kontener wywołuje metodę **onMessage** komponentu typu Message-Driven
2) Metoda **onMessage** normalnie rzutuje otrzymaną wiadomość do jednego z pięciu typów komunikatów JMS i obsługuje je zgodnie z logiką biznesową aplikacji. **Metoda onMessage może wywołać metody pomocnicze lub może wywołać metodę komponentu typu Session Bean** do przetwarzania informacji w wiadomości lub **może zapisać je w bazie danych**.
3) Wiadomość może być dostarczona do komponentu typu Message-Driven w kontekście transakcji, więc wszystkie operacje w metodzie **onMessage są częścią jednej transakcji**. Jeśli przetwarzanie wiadomości zostanie wycofane, wiadomość zostanie zwrócona.

2.3.2. Kiedy stoduje się komponenty typu Message-Driven?

Aby odbierać wiadomości w sposób asynchroniczny, ponieważ:

1) Aby uniknąć uruchamiania zasobów serwera, czyli aby nie używać blokowania synchronicznego odbioru wiadomości po stronie serwera;
2) Komunikaty JMS powinny być wysyłane lub odbierane asynchronicznie.

Komponenty typu Session Bean mogą wysyłać i odbierać wiadomości **JMS tylko synchronicznie.**

4. Dostęp do komponentów typu Enterprise (z wyłączeniem komponentów typu Message-Driven)

Warstwa klienta uzyskuje dostęp do komponentów EJB

1) **bez interfejsu biznesowego** – wtedy może wywołać wszystkie publiczne metody komponentu EJB
2) **przez interfejs biznesowy** – wtedy może wywołać tylko te metody, które implementuje komponent, a są zadeklarowane w interfejsie biznesowym.

4. Dostęp do komponentów typu Enterprise (z wyłączeniem komponentów typu Message-Driven) (cd)

1) Mechanizm dostępu oparty na dobrze zaprojektowanych interfejsach biznesowych lub braku interfejsu upraszczają opracowywanie i utrzymanie aplikacji Java EE.
2) Taki mechanizm dostępu ukrywa przed warstwą klienta wszelkie zawiłości w warstwie biznesowej i pozwala modyfikować definicje metod bez wpływu na kod warstwy klienta.
3) Jeśli jednak zostanie zmodyfikowana nazwa metody w komponencie EJB, to wymaga zmiany kodu w warstwie klienta.
4) Komponent typu EJB może mieć więcej niż jeden interfejs biznesowy. Komponenty typu EJB powinny, ale nie muszą, implementować wszystkich interfejsów biznesowych.

4.1 Sposób użycia komponentu typu Enterprise Beans w warstwie klienta

Warstwa klienta uzyskuje odwołanie komponentu EJB:

1) poprzez wstrzyknięcie zależności (**Dependency Injection**) używając adnotacji języka programowania Java za pośrednictwem klasy javax.ejb.EJB np. w warstwie klienta w obrębie środowiska serwera JavaEE, w aplikacjach JSF, JAX-RS Web Services.
2) za pomocą operacji **lookup** realizowanej w JNDI (**Java Naming and Directory Interface**) - aplikacje działające poza środowiskiem serwera Java EE

4.1.1. Trzy przestrzenie nazw w procesie lookup technologii JNDI

1) **java:global** – ścieżka wyszukiwania zdalnego komponentu EJB

java:global [/nazwa aplikacji] /nazwa modulu/

nazwa komponentu EJB[/nazwa interfejsu]

2) **java:module –** ścieżka wyszukiwania komponentu typu EJB w tym samym module

java:module/nazwa komponentu EJB / [nazwa interfejsu]

2) **java:app –** ścieżka wyszukiwania komponentu EJB w tej samej aplikacji, spakowanego w pliku EAR

java:app[/nazwa modulu]/nazwa komponentu EJB

[/nazwa interfejsu]

Nazwa interfejsu biznesowego jest używana wtedy, gdy komponent EJB implementuje więcej niż jeden interfejs.

4.2 Kiedy należy stosować zdalny, kiedy lokalny, a kiedy jako usługa internetowa

dostęp do komponentu typu EJB

1) **silne lub luźne logiczne sprzężenie komponentów EJB** - przy silnym powiązaniu lepiej stosować **lokalny dostęp** między komponentami (np jeden obsługuje sprzedaż, a drugi wysyła maile potwierdzające sprzedaż)
2) **typ klienta:** instancje warstwy klienta umieszczane na innych maszynach niż komponent typu EJB dostarczający usługi biznesowe – **lepszy zdalny dostęp**
3) **dystrybucja komponentów:** aplikacje Java EE są skalowalne, ponieważ ich komponenty po stronie serwera może być rozłożone na wielu maszynach. Wtedy konieczny jest **dostęp zdalny** pomiędzy komponentami EJB
4) **Wydajność:** Ze względu na takie czynniki, jak opóźnienia w sieci, **połączenia zdalne mogą być wolniejsze** od połączeń lokalnych

Bardziej elastyczne **są połączenia zdalne.** Nie można jednocześnie definiować interfejsu biznesowego jako lokalnego i zdalnego.

4.3. Lokalna warstwa klienta

Charakterystyka:

1) Musi działać w tej samej aplikacji, w której działa komponent typu EJB
2) Musi być komponentem internetowym lub innym komponentem EJB
3) Dla lokalnej warstwy klienta dostęp do komponentu EJB nie jest jawny.

Brak interfejsu biznesowego w dostępie do metod komponentu EJB oznacza **lokalny dostęp** do **pulicznych metod komponentu**.

Lokalny interfejs biznesowy oznacza implementację metod i cykl życia tych metod.

Brak adnotacji @Remote, @Local oznacza domyślnie **lokalny interfejs biznesowy.**

4.3. Lokalna warstwa klienta – definicje komponentu EJB (cd)

Przykłady:

1) Brak interfejsu biznesowego– definicja komponentu

```
@Session
public class MyBean { ... }
```

2) Definicja lokalnego interfejsu biznesowego:

```
@Local
public interface InterfaceName { ... }
@Session
public class MyBean implements InterfaceName { ... }
```

3) Specyfikacja interjesju biznesowego w nazwie komponentu

```
@Local(InterfaceName.class)
public class MyBean implements InterfaceName { ... }
```

4.3.1. Lokalna warstwa klienta – dostęp do komponentu typu EJB przy brak interfejsu biznesowego

Definicja komponentu EJB

```
@Session
public class MyBean { ... }
```

1) Dostęp do komponentu MyBean za pomocą adnotacji javax.ejb.EJB z lokalnej warstwy klienta

```
@EJB
 MyBean myBean;
```

2) Dostęp do komponentu EJB za pomocą **JNDI**, **lookup** (javax.naming.InitialContext)

```
MyBean myBean = (MyBean)
        InitialContext.lookup   (”java:module/MyBean”)
```

4.3.2. Lokalna warstwa klienta – dostęp do komponentu typu EJB z definicją interfejsu biznesowego

Definicja komponentu EJB

```
@Local
public interface InterfaceName { ... }
@Session
public class MyBean implements InterfaceName { ... }
```

1) Dostęp do komponentu MyBean za pomocą adnotacji javax.ejb.EJB z lokalnej warstwy klienta

```
@EJB
 MyBean myBean;
```

2) Dostęp do komponentu EJB za pomocą **JNDI**, **lookup** (javax.naming.InitialContext)

```
MyBean myBean = (MyBean)
        InitialContext.lookup (”java:module/MyBean”)
```

4.4. Zdalni klienci

Charakterystyka:

1) Działa na innej maszynie, z wykorzystaniem innej maszyny wirtualnej Java (może być również ta sama maszyna JVM)
2) Może to być komponent internetowy, aplikacja klienta, lub inny komponent typu EJB
3) Dla klienta zdalnego dostęp do komponentu typu EJB jest „przezroczysty”
4) Komponent EJB musi implementować interfejs biznesowy – w przeciwnym wypadku warstwa klienta nie ma dostępu do takkiego komponentu typu EJB

4.4. Zdalni klienci (cd)

Definicja komponentu EJB dla zdalnego klienta:

@Remote

public interface InterfaceName { ... }

@Remote(InterfaceName.class)

public class MyBean implements InterfaceName { ... }

1) Dostęp do komponentu EJB ze zdalnej warstwy klienta za pomocą adnotacji javax.ejb.EJB

   @EJB

   MyBean myBean;

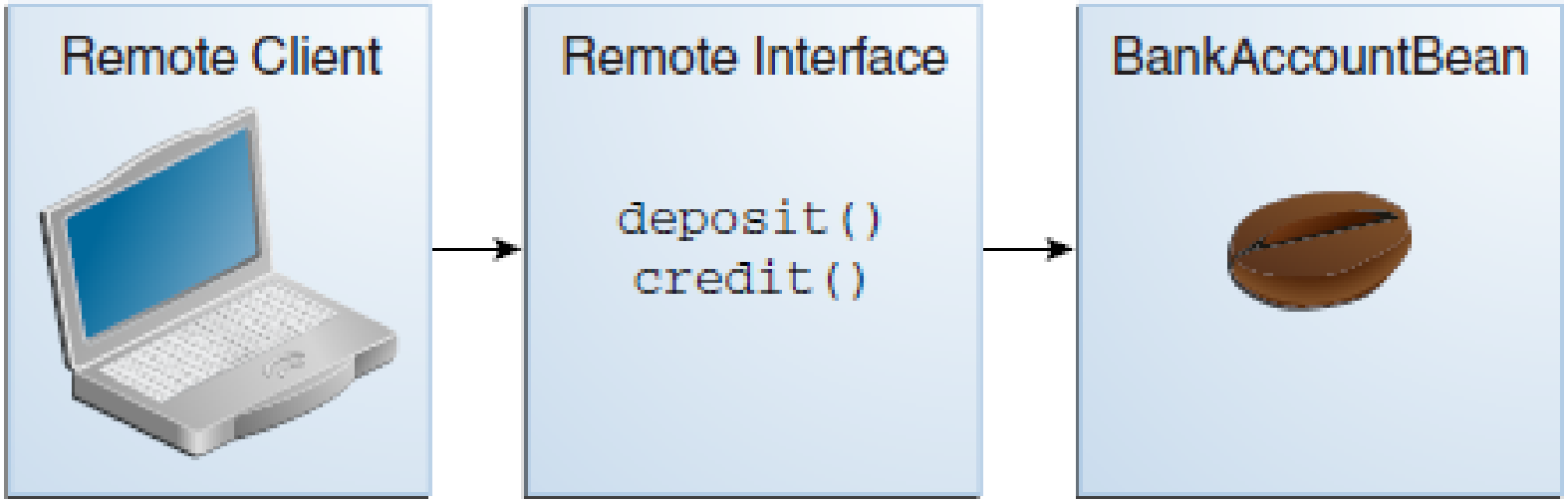
2) Dostęp do komponentu EJB za pomocą **JNDI**, **lookup** (javax.naming.InitialContext)

   MyBean myBean = (MyBean)

   InitialContext.lookup (”java:global/MyApp/MyBean”);

4.4. Zdalni klienci (cd)

Kontrola odwołań zdalnej warstwy klienta do komponentu typu EJB za pomocą zdalnego interfejsu biznesowego



4.5. Klienci usługi internetowej (Web Service)

Tworzenie usługi internetowej:

1) Warstwa klienta ma dostęp do usługi internetowej wykonanej za pomocą technologii JAX-WS
2) Warstwa klienta usługi internetowej wywołuje metody biznesowe bezstanowego (stateless) komponentu typu Session Bean z wykorzystaniem protokołów (SOAP, HTTP, WSDL).
3) Warstwą klienta usługi internetowej moga być: komponent internetowy oraz komponent typu EJB. Umożliwia to integrację aplikacji Java EE z usługami internetowymi.
4) Warstwa klienta usługi internetowej uzyskuje dostęp do bezstanowego komponentu Session Bean za pomocą klasy implementującej tzw. „endpoint” komponentu usługi internetowej. Można wywołać metody usługi internetowej zdefiniowane z adnotacją @**WebMethod.**

4.5. Parametry zdalnych metod dostępu do komponentów typu EJB

1) **Separacja -** parametry metod zdalnych są odseparowane od parametrów metod wywoływanych lokalnie. Przy zdalnych wywołaniach warstwa klienta i komponent EJB **operują na różnych kopiach parametrów obiektowych.** W przypadku, gdy warstwa klienta zmienia wartość tych parametrów, kopia ich po stronie komponentu EJB nie jest zmieniana.

   Podobnie jest podczas wywołań metod usług internetowych.

   Podczas lokalnych wywołań obie strony modyfikują te same parametry. To powoduje, że korzystniejsze są wywołania zdalne (Remote).

2) **Ziarnistość używanych danych liście parametrów lub zwracanych przez instrukcję return w** metodach komponentów EJB – lepiej przesyłać mniej obiektów, ale z większą liczbą danych (gruboziarniste).

5. Zawartość komponentu typu EJB

1) **Klasa typu Enterprise bean**: implementuje metody biznesowe i metody cyklu życia typu callback.
2) **Biznesowe interfejsy (interface)**: definiuje metody biznesowe implementowe przez komponenty typu EJB, niezbędne przy zdalnym dostępie do metod komponentu typu EJB
3) **Klasy pomocnicze**: klasy potrzebne do definicji metod klas komponentów typu EJB np do obsługi wyjątków i inne klasy pomocnicze

.

Pliki te spakowane są w formacie jJAR (komponenty typu EJB), WAR (komponenty typu internetowego)

6. Konwencja nazw dla komponentów typu EJB

Pozycja	Składnia	Przykład
Nazwa komponentu typu EJB	nazwaBean	AccountBean
Nazwa klasy typu EJB	nazwaBean	AccountBean
Interfejs biznesowy	nazwa	Account

7. Cykl życia komponentów typu EJB

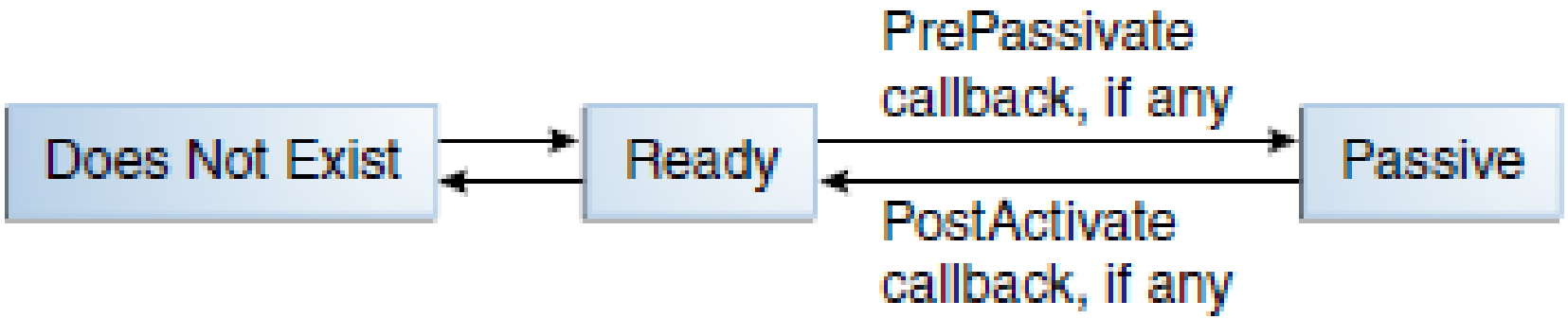
Każdy z komponentów EJB ma różny cykl życia i czas życia:

- Stateful
- Stateless
- Singleton
- message-driven

7.1 Cykl życia stanowych (Stateful) komponentów typu Session Bean

Zadania kontenera EJB:

1. Tworzy komponent
2. Dependency injection, jeśli są
3. Metoda PostConstruct callback, jeśli jest
4. Metoda Init, lub ejbCreate<METHOD>, jeśli są



1. Usuwa komponent
2. Metoda PreDestroy callback, jeśli jest

7.1 Cykl życia stanowych (stateful) komponentów typu Session Bean (cd)

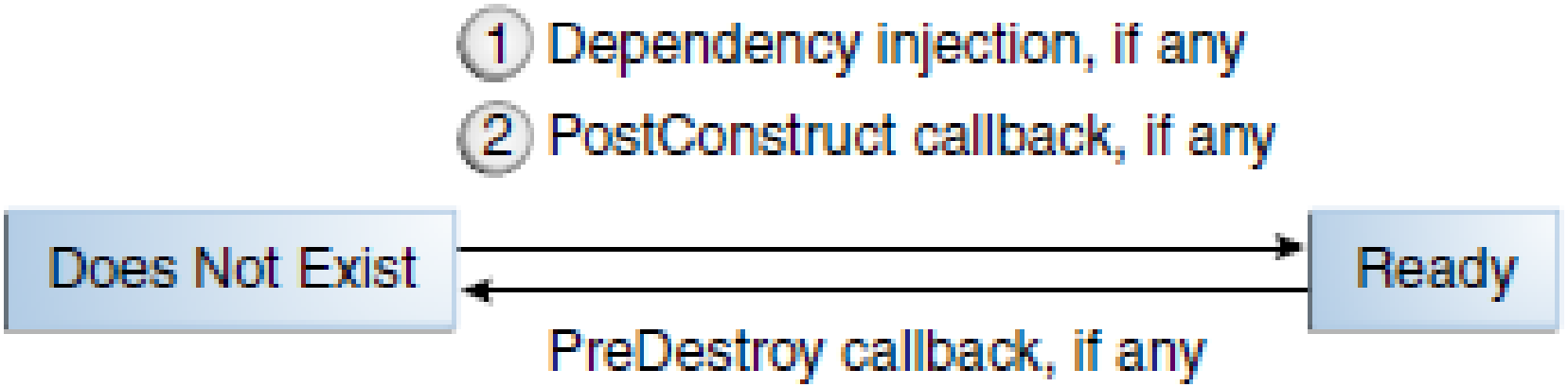
Warstwa klienta inicjuje cykl życia stanowego komponentu typu Session Bean. Kontener EJB realizuje „wstrzyknięcie” referencji do komponentu EJB i wywołuje metodę z adnotacją @PostConstruct, jeśli jest. Po takim wstępie z warstwy klienta można wywołać metody z komponentu EJB.

W stanie Ready kontener EJB decyduje, czy dokonać pasywacji lub dezaktywacji (algorytm: najmniej używany jest przesuwany do pamięci pomocniczej) – wywołując metody z adnotacjami @PrePassivate (jeśli jest). Jeśli warstwa klienta wywoła wtedy metodę komponentu EJB, wywołana jest przez kontener metoda @PostActive (jeśli jest).

Podczas zakończenia działania warstwy klienta, wywołana jest przez kontener metoda z adnotacją @PreDestroy (jeśi jest). Wtedy komponent EJB jest usuwany z pamięci.

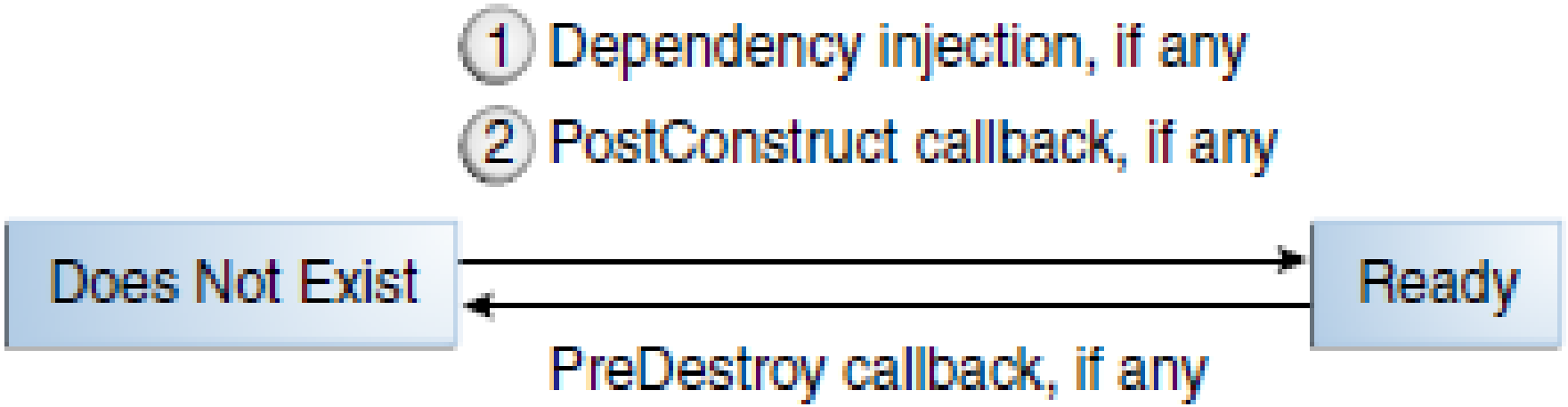
7.2. Cykl życia bezstanowych (Stateless) komponentów typu Session Bean

Ponieważ bezstanowy komponent typu Session Bean nigdy nie przechodzi w stan pasywny, cykl życia składa sie tylko z dwóch stanów: **nonexistent and ready** podczas wywoływania metod przez warstwę klienta.



7.3. Cykl życia komponentów Singleton typu Session Bean

Ponieważ bezstanowy komponent typu Session Bean nigdy nie przechodzi w stan pasywny, cykl życia składa sie tylko z dwóch stanów: **nonexistent and ready** podczas wywoływania metod przez warstwę klienta. Singleton startuje razem z aplikacją, jeśli ma adnotację @**StartUp** i inicjuje aplikację za pomocą metody @PostConstruct (jeśli jest) i „czyści” za pomocą metody @PreDestroy (jeśli jest).



7.4 Cykl życia komponentu typu Message-Driven Bean

Kontener EJB tworzy pulę komponentów typu Message-Driven wykonując podane zadania:

1) Wykonuje wszystkie „wstrzyknięcia” , jeśli są
2) Kontener wywołuje metodę @PostConstruct, jeśli jest