# Arquitetura JEE
# Introdução à Camada de Negócios: Enterprise Java Beans (EJB)

OBJETOS DISTRIBUIDOS

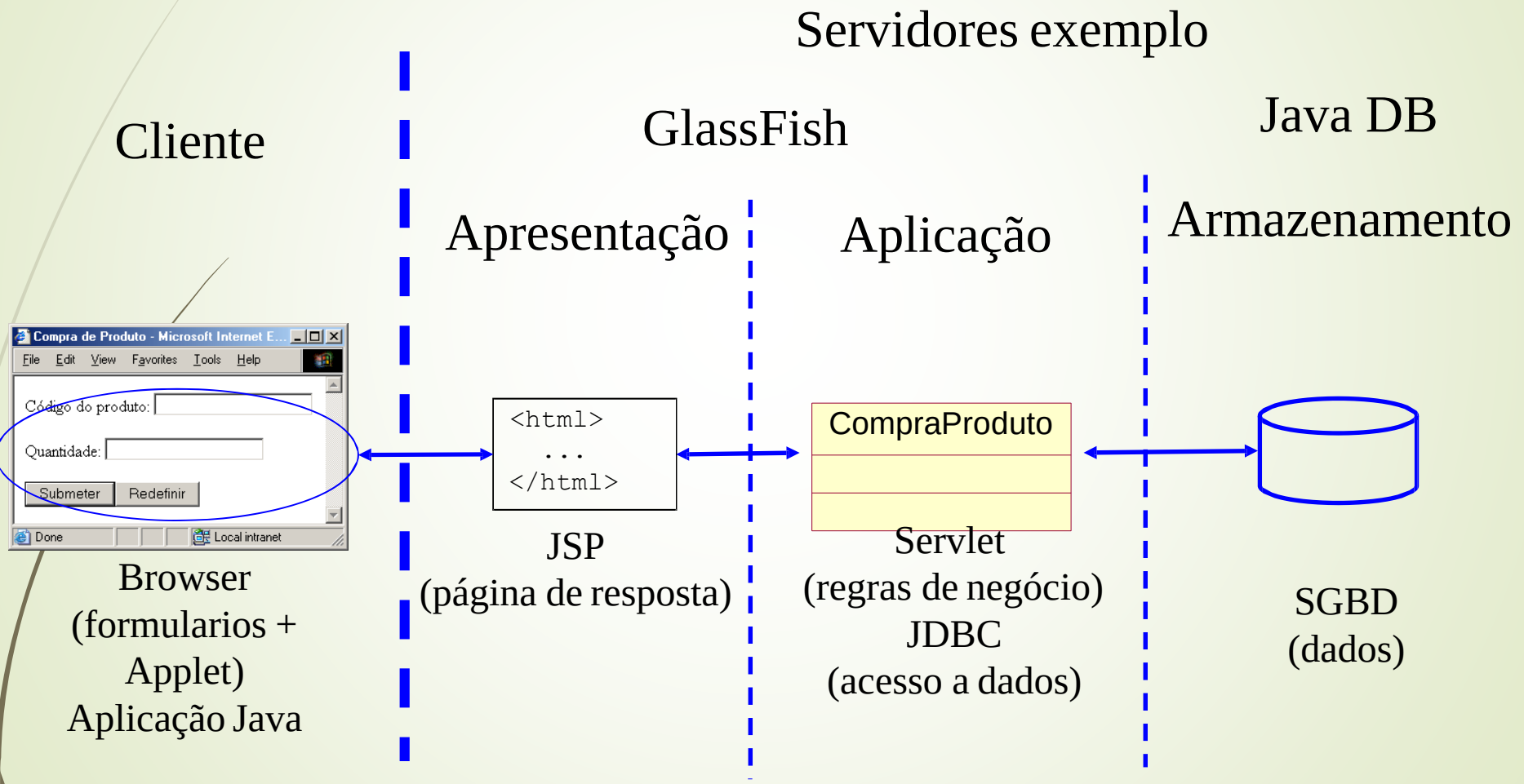
# Agenda

- **Arquiteturas Web em Java**
- Arquitetura Java EE
- Introdução a Enterprise Java Beans (EJB)
- Discussão: POO X Programação Orientada a Componentes (POC)
- Quando usar EJBs?

# Arquiteturas Web em Java

- Java permite a adoção de diferentes arquiteturas web
- Dentre as mais famosas estão
  - Model2 (mais simples)
  - Java EE (mais complexa)

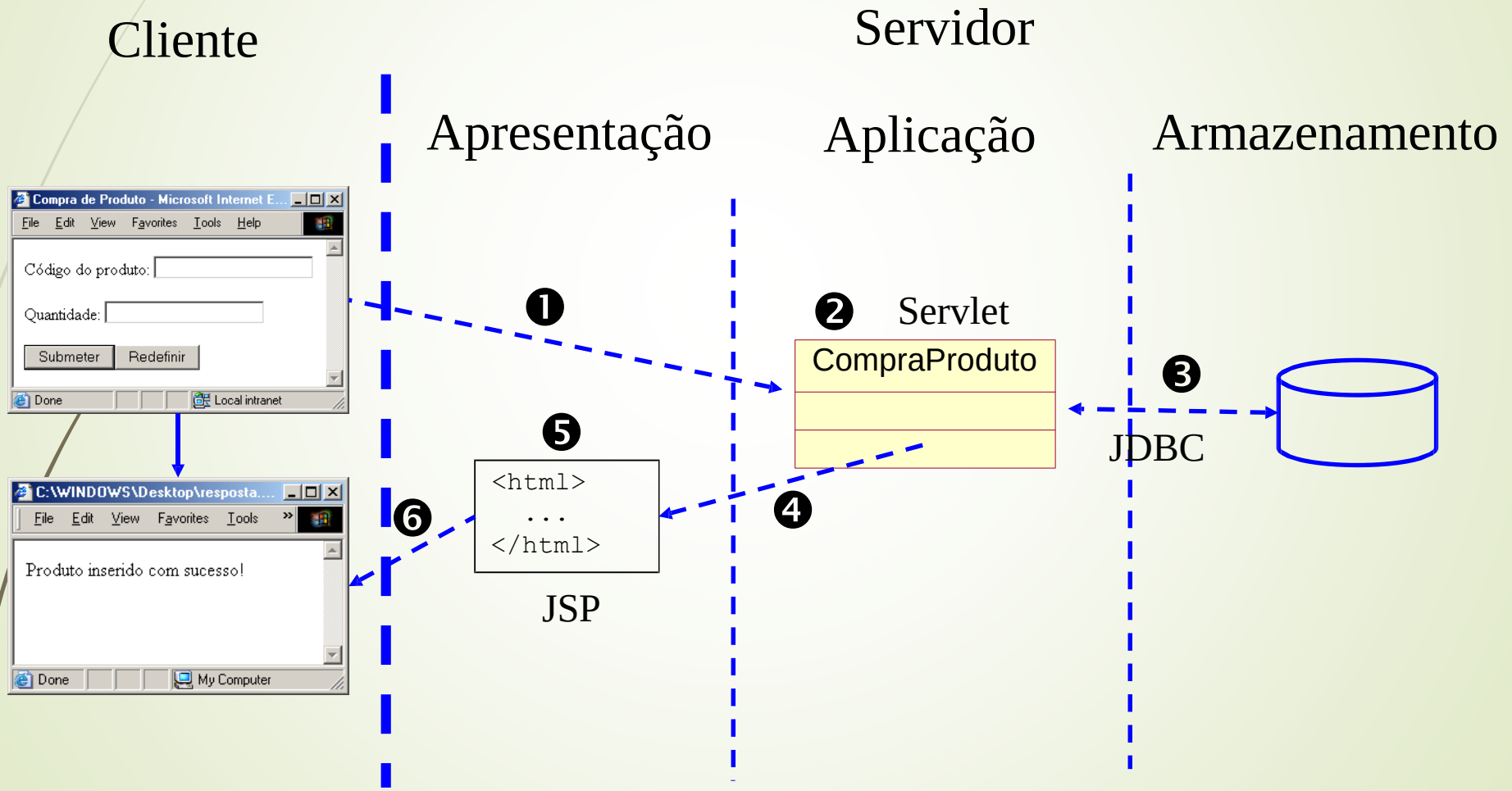
# Arquitetura Model2 em Java



# Arquitetura Model2 em Java (elementos fundamentais)

- JSP
  - Páginas HTML com código Java embutido
- Servlet
  - Classes Java que rodam em servidores
- JDBC
  - API de acesso a banco de dados em Java

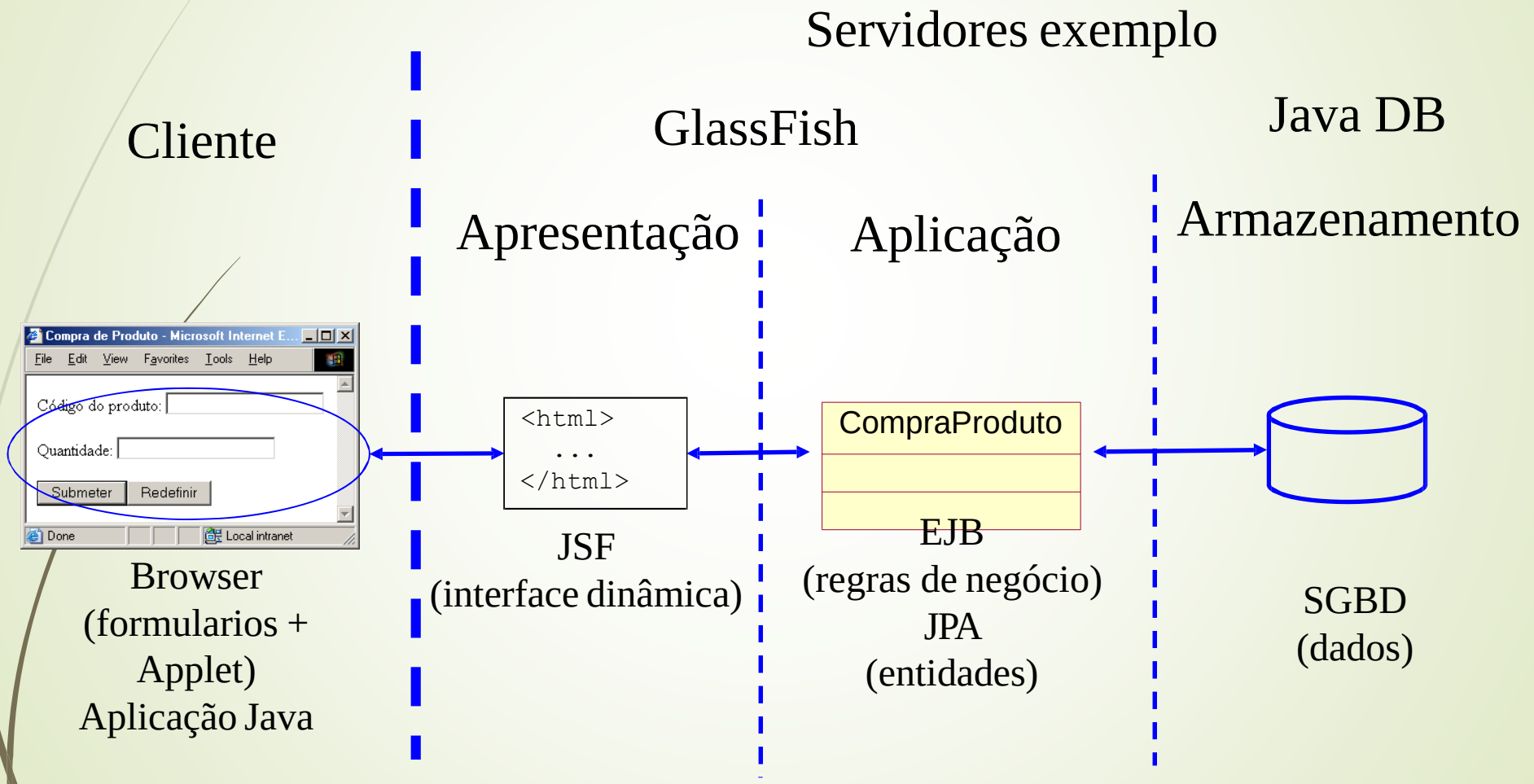
# Arquitetura Model2 em Java (cenário típico)



# Arquitetura Model2 em Java (cenário típico)

1. Cliente solicita um Servlet usualmente após o preenchimento de um formulário HTML
2. Servidor interpreta o Servlet na camada de aplicação
3. Se necessário, a camada de aplicação se comunica com a camada de armazenamento através de JDBC
4. Camada de aplicação redireciona o fluxo para a camada de apresentação
5. Servidor constrói uma página de resposta usando JSP
6. Servidor retorna a página de resposta.

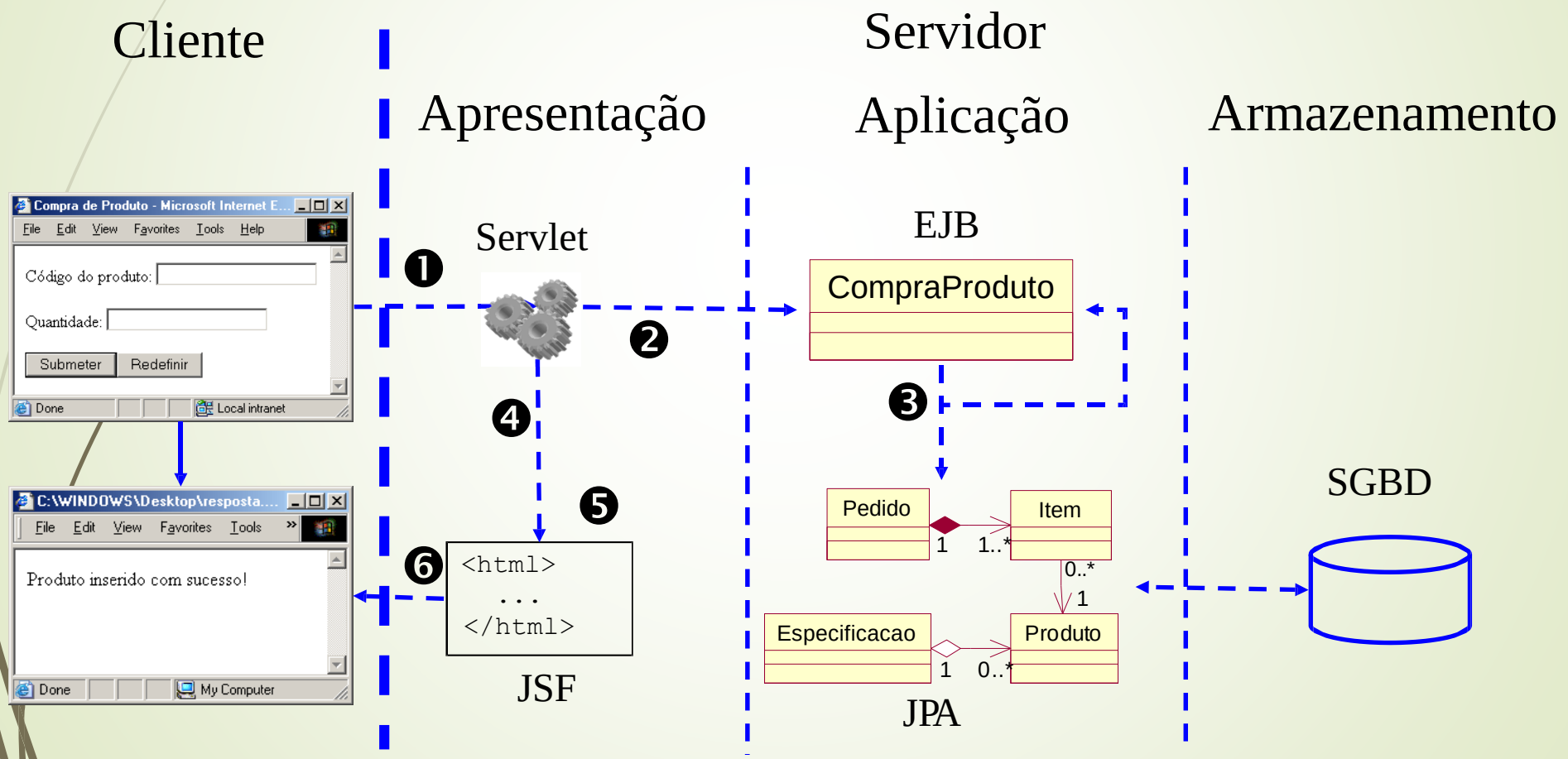
# Arquitetura Java EE



# Arquitetura Java EE (elementos fundamentais)

- JSF
  - Framework de apresentação que faz uso disciplinado de Servlet e JSP
- EJB
  - Componentes de negócio
- JPA
  - Entidades persistentes

# Arquitetura Java EE (cenário típico)



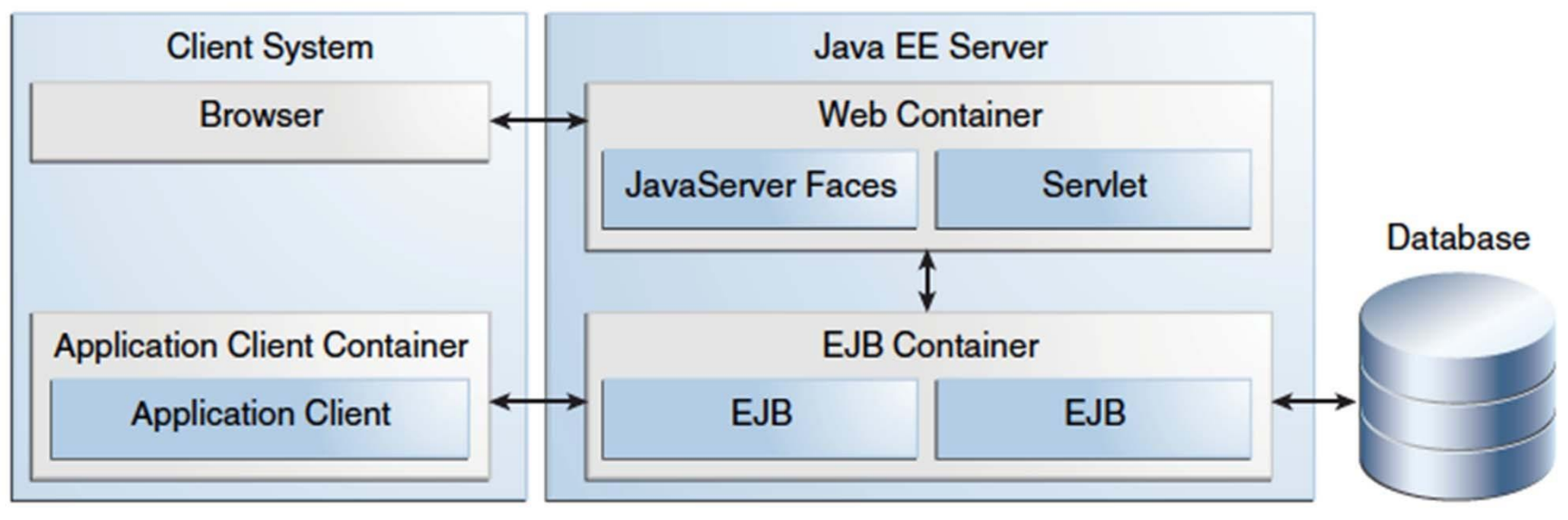
# Arquitetura Java EE (cenário típico)

1. Cliente solicita um Servlet usualmente após o preenchimento de um formulário HTML
2. Servidor interpreta o Servlet e redireciona o fluxo para um EJB na camada de aplicação
3. Se necessário, a camada de aplicação faz uso de outros EJBs ou se comunica com a camada de armazenamento através de entidades JPA
4. Camada de aplicação redireciona o fluxo para a camada de apresentação
5. Servidor constrói uma página de resposta usando JSF
6. Servidor retorna a página de resposta

# Agenda

- ✓ Arquiteturas Web em Java
- ➢ **Arquitetura Java EE**
- Introdução a Enterprise Java Beans (EJB)
- Discussão: POO X Programação Orientada a Componentes (POC)
- Quando usar EJBs?

# Arquitetura Java EE




Fonte: livro Java EE 7 Tutorial

# Arquitetura Java EE

- Arquitetura para aplicações distribuídas e multi-camadas;
  - Baseada na linguagem Java: Independente de plataforma;
  - É uma especificação: Independente de vendedor.
- Arquitetura de componentes.

# A Camada Cliente

- A camada cliente está na máquina do usuário e é formada basicamente por clientes *web* ou aplicações Java no cliente.
- Um cliente *web* consiste de *applets*, páginas estáticas, páginas dinâmicas geradas pelo servidor e de um *browser* que exibe a página recebida do servidor.
- A arquitetura defende o uso clientes “magros” (*thin clients*) e a separação de lógica de negócio da lógica de apresentação.

# A Camada Intermediária *Middle Tier* (de Negócio)

- A camada intermediária contém o servidor JEE.
- O servidor contém e gerencia os componentes *web* e os objetos de negócio por meio dos *containers.*
  - Os elementos *web* (*servlets,* páginas *jsp*) processam dinamicamente as requisições dos usuários, construindo as páginas de resposta para a camada cliente.
  - Os objetos de negócio (*enterprise beans*) implementam a lógica de um domínio de negócio.

# A Camada de Dados

- A camada de dados permite a persistência dos dados do negócio.
  - Sistemas de banco de dados, ERP, transações de mainframe e outros sistemas de informação legados.

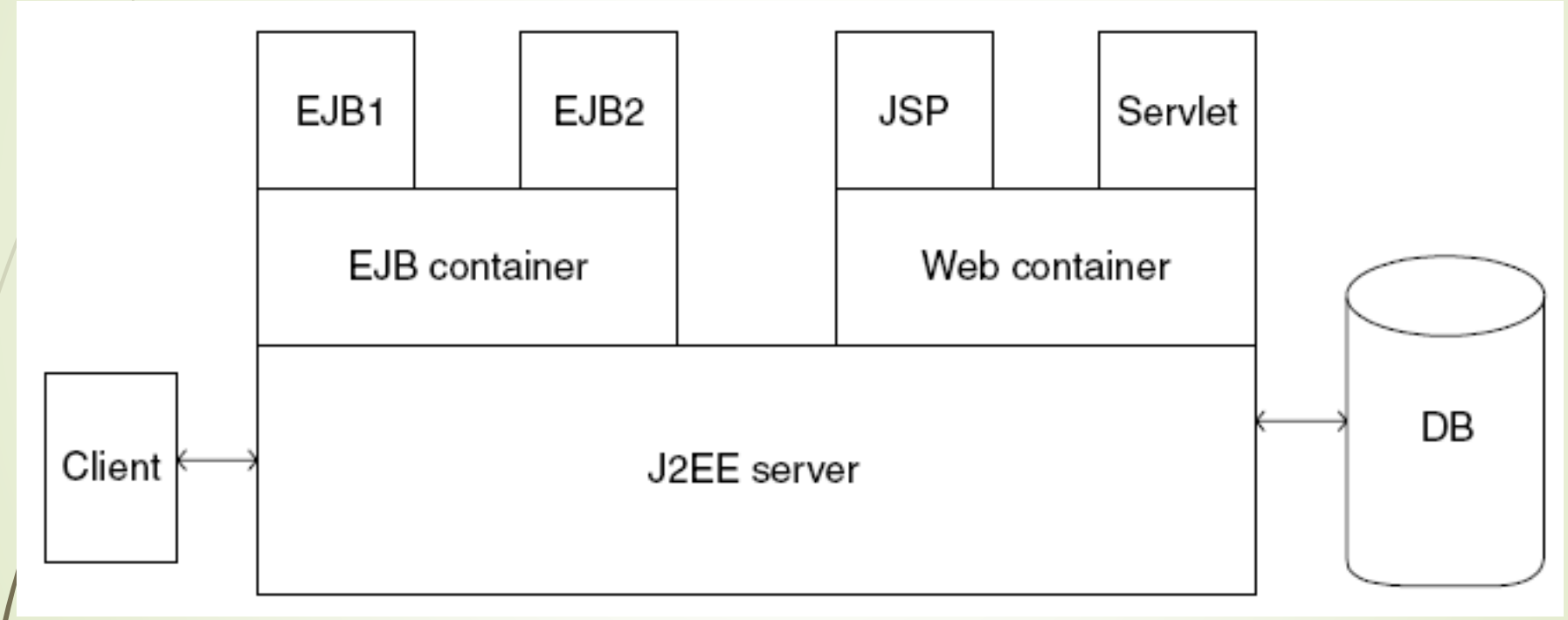
# Containers

- Infraestrutura capaz de oferecer serviços básicos para códigos Java
  - Segurança
  - Transação
  - Lookup e injeção de dependências e recursos
  - Conectividade remota
  - Gestão do ciclo de vida

# Containers

- Container Web
  - Interface entre componentes web (Servlets, JSP ou JSF) e o servidor Java EE
- Container EJB
  - Interface entre componentes EJB e o servidor Java EE e/ou aplicações que dele possam fazer uso
    - Isola o componente EJB de acesso direto por seus clientes
  - Ambiente *runtime* que controla os componentes EJB durante seu tempo de vida, provendo todos os serviços de gerenciamento necessários
  - Antes de ser executado, um componente EJB deve ser implantado (*deployed*) no seu *container* específico

# O Servidor e os Containers



# Agenda

- ✓ Arquiteturas Web em Java
- ✓ Arquitetura Java EE
- ➢ **Introdução a Enterprise Java Beans (EJB)**
- Discussão: POO X Programação Orientada a Componentes (POC)
- Quando usar EJBs?

# Enterprise JavaBeans

- Tanto a camada de apresentação (JSF, Swing etc) quanto a camada de persistência (JPA) não são apropriadas para desempenharem a lógica de negócio
- Muitas vezes, para realizar tarefas de negócio complexas é necessário o acesso a outros componentes e até mesmo serviços externos com independência de interface gráfica
  - Por isto, as camadas de apresentação e persistência não devem conter a lógica de negócio!
- EJBs são componentes do lado do servidor que encapsulam a lógica de negócio e cuidam de aspectos como transação, escalabilidade e segurança
  - Estão integrados com serviços como mensagens, agendamento de tarefas e web services

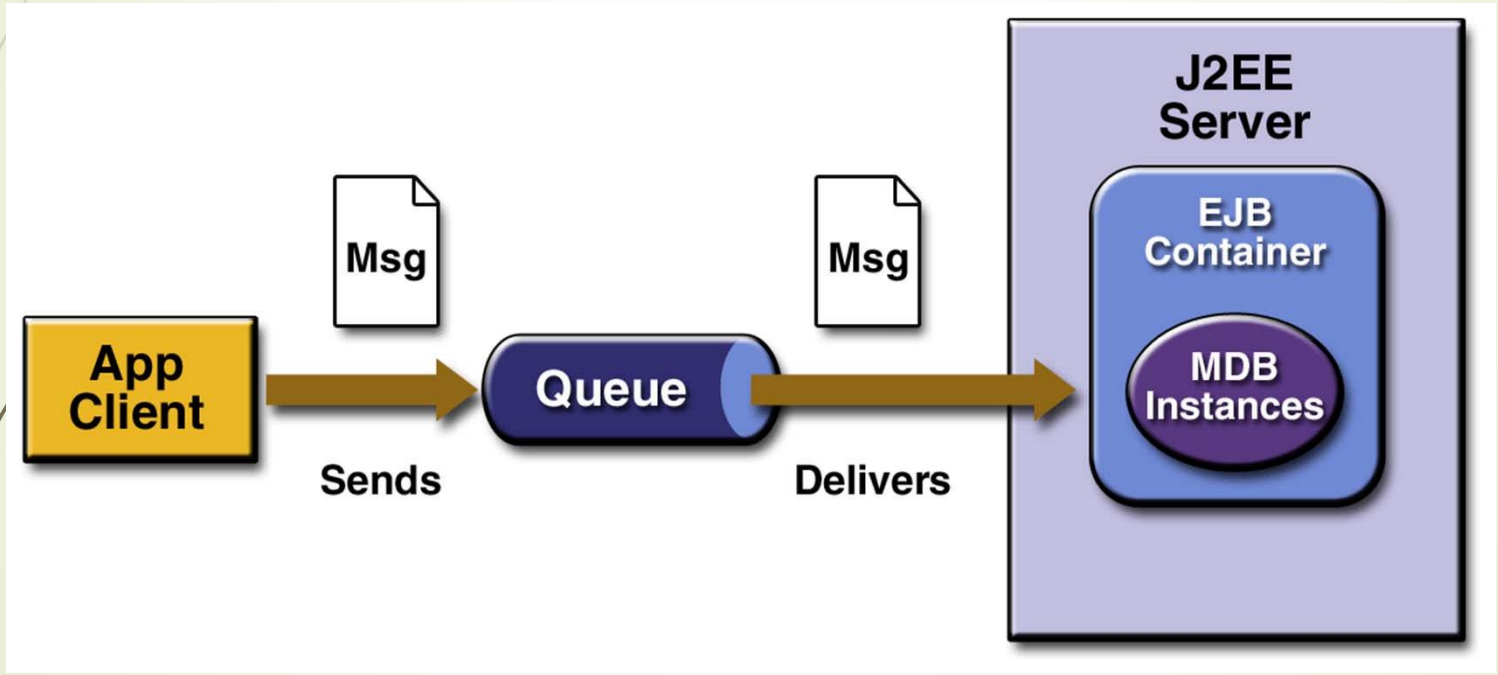
# Tipos de EJB

- *Session beans* são usados para encapsular regras de negócio de alto nível, principalmente aquelas envolvendo um fluxo ou um processo de trabalho. Podem ser de três tipos:
  - *Stateless* não mantém o estado da conversação entre métodos e qualquer instância pode ser usada por qualquer cliente
  - *Stateful* mantém o estado da conversação entre métodos, o qual precisa ser mantido para um único usuário
  - *Singleton* é uma instância única que é compartilhada entre os diferentes clientes e suporta acesso concorrente

# Tipos de EJB

- *Message-driven beans* são consumidores assíncronos de mensagens de filas do JMS (Java Message Service)
- *EJB timer service* é a forma padrão Java EE para agendamento de tarefas
- Os EJBs podem ainda ser *endpoint* para Web Services

# Message Driven Beans



# A Anatomia de um EJB

- *Session beans* encapsulam lógica de negócio, são transacionais, utilizam-se de um container que provê *pooling*, *multithreading*, segurança etc
  - Quais artefatos são necessários para criar um componente “poderoso” como este?
    - Apenas uma anotação!

```
@Stateless
public class LivroEJB {
    @PersistenceContext(unitName = “exemplo")
    private EntityManager em;

    public Livro buscaLivroPorId(Long id) {
        return em.find(Livro.class, id);
    }
    public Livro persisteLivro(Livro livro) {
        em.persist(livro);
        return livro;
    }
}
```

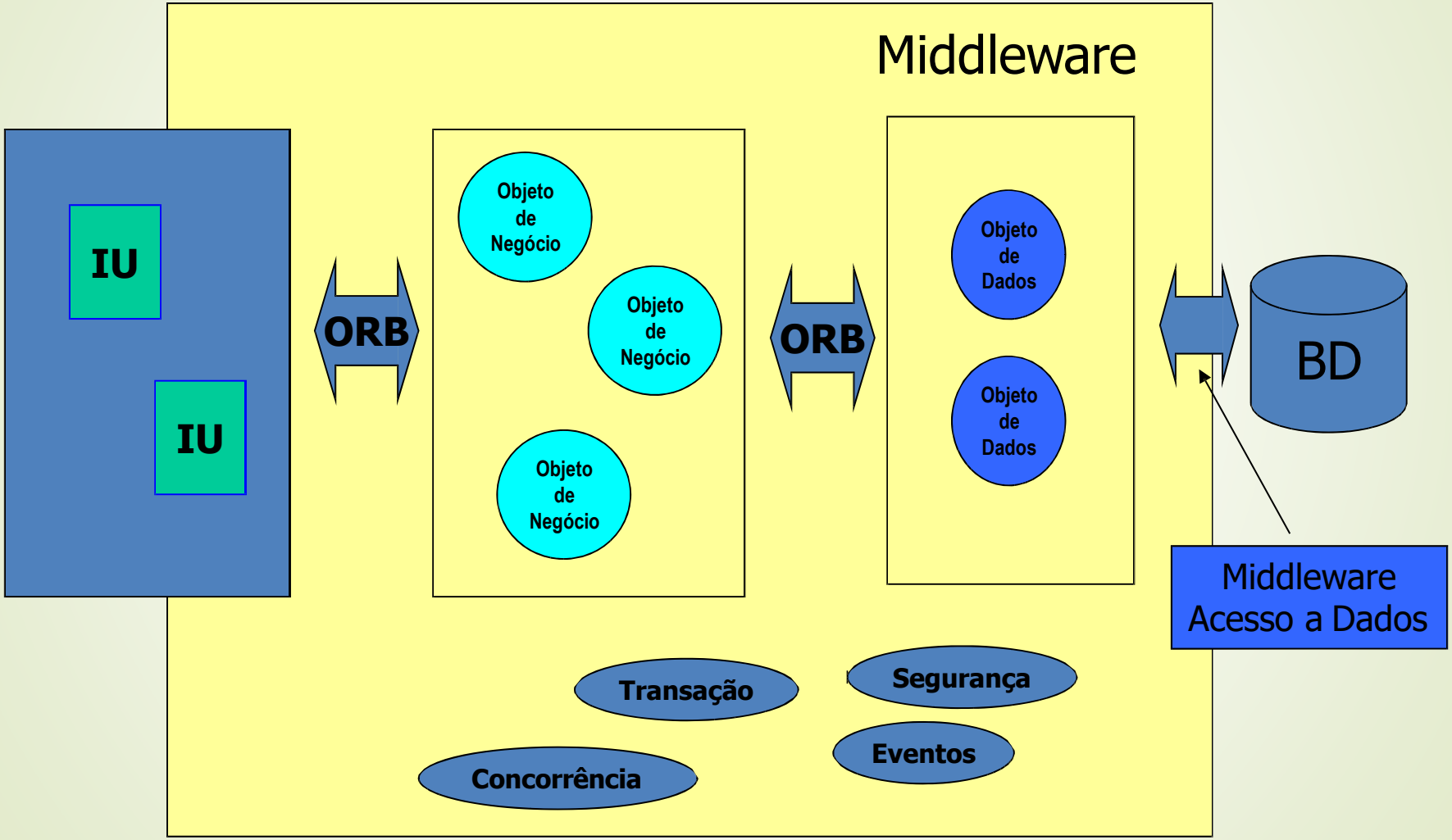
# Um Cliente de um EJB

```
public class Main {
    @EJB
    private static LivroEJBRemote livroEJB;

    public static void main(String[] args) {
            Livro livro = new Livro();
            livro.setTitulo("O Guia do Mochileiro das Galáxias");
            livro.setPreco(38.5F);
            livro.setDescricao("Humor e ficção científica");
            livro.setNumPaginas(380);

            livroEJB.persisteLivro(livro);
    }
}
```

# Clientes e o EJB Container

- EJBs são objetos gerenciados e, por isso, precisam de um container para serem executados
- Containers fornecem diversos serviços como injeção de dependência, gerenciamento de transações, etc.
- Quando um **cliente EJB** invoca um EJB, ele não opera diretamente com uma instância deste EJB, mas com um *proxy* para aquela instância
  - Toda vez que um método é invocado, a chamada é "roteada" através do container, provendo alguns serviços em nome da instância do bean EJB
    - Tudo isto é transparente para para o cliente, da sua criação até a destruição, um EJB reside no container

# Clientes e o EJB Container



# Clientes e o EJB Container Injeção de Dependência e JNDI

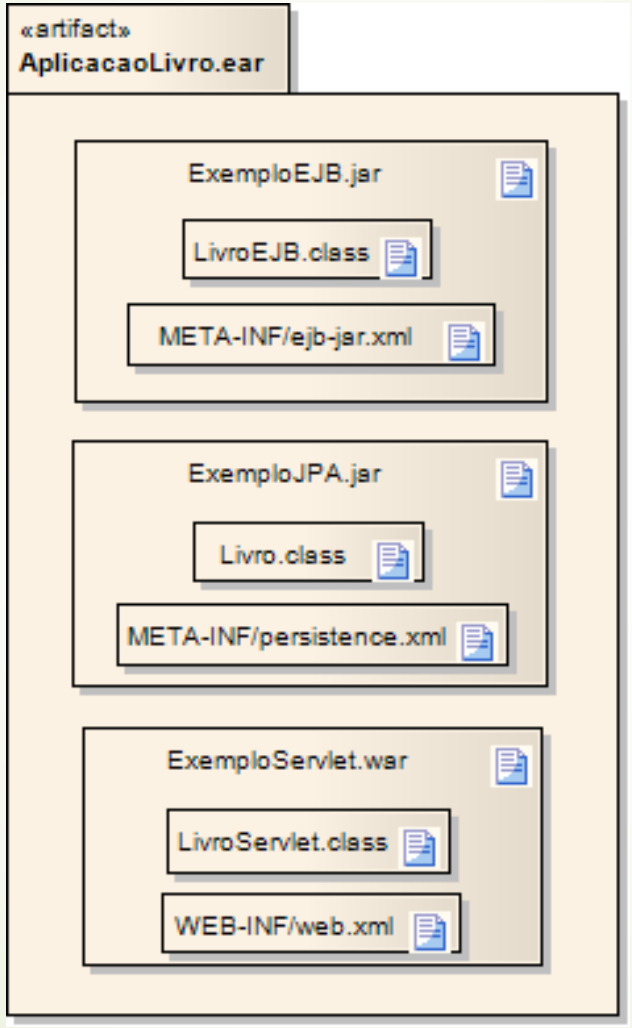
- A injeção de dependência é um mecanismo útil (e transparente) para a instanciação de objetos (no exemplo visto, EJBs)
- O Java Name and Directory Interface (JNDI) é uma alternativa à injeção de dependência onde objetos podem ser invocado remotamente através de um **nome único**
  - O JNDI é definido no Java SE
- Convenção de nome para EJB:
  - java:global[/<app-name>]/<module-name>/<bean-name>
- Usando o JNDI para invocar um EJB

```
Context ctx = new InitialContext();
LivroEJB livroEJB = (LivroEJB)  ctx.lookup("java:global/ExemploEJB/LivroEJB");
```

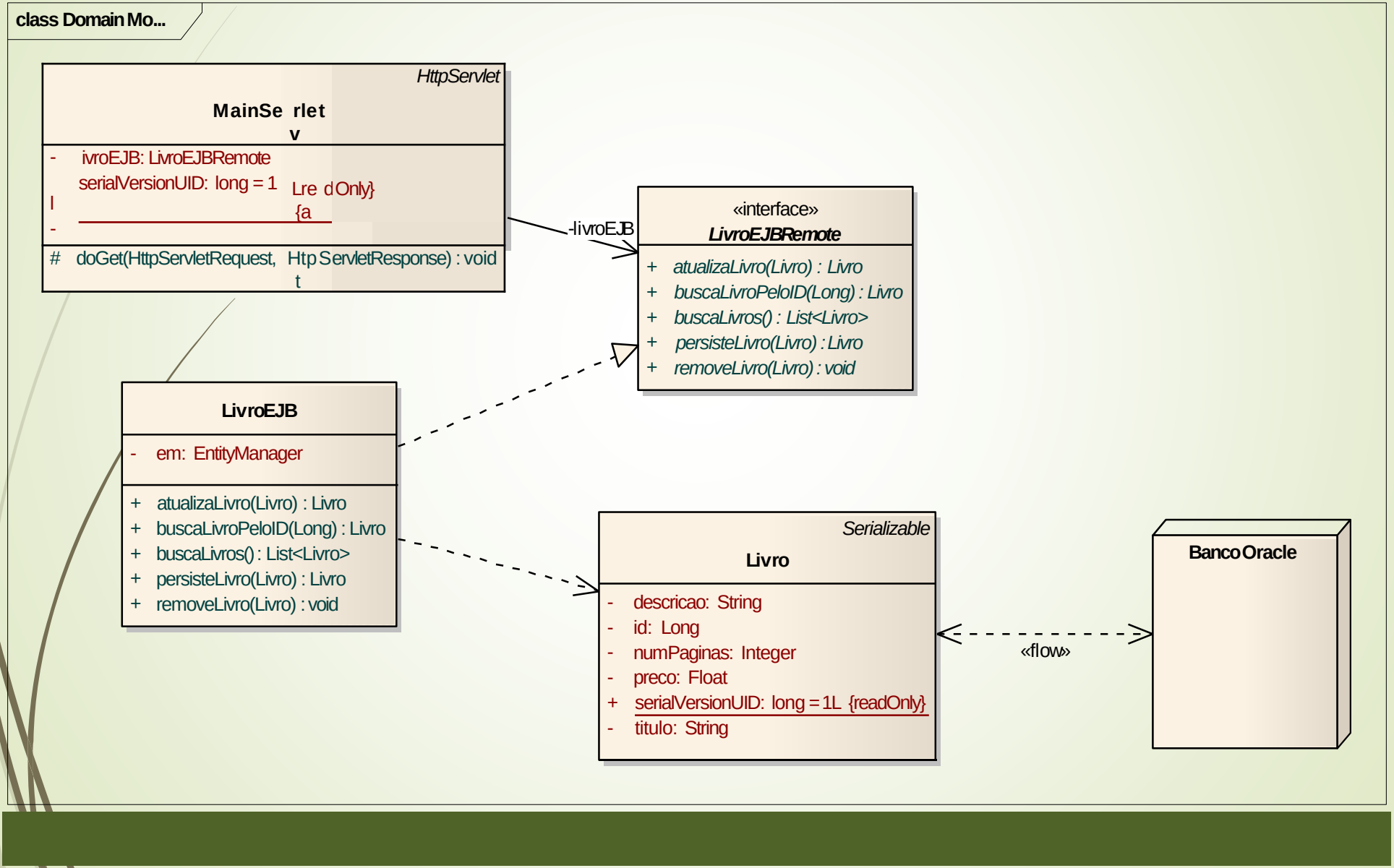
# Clientes EJB

- Como visto, EJBs são componentes remotos gerenciados por um container
- Na grande maioria dos casos, os clientes ou são aplicações Web ou são aplicações Java SE
- Aplicações Java SE acessam EJBs **via JNDI**
- Aplicações Web, por já estarem no container, podem **usar anotações** (injeção de dependência)

# Empacotamento



# Exemplo de Aplicação Web com EJB

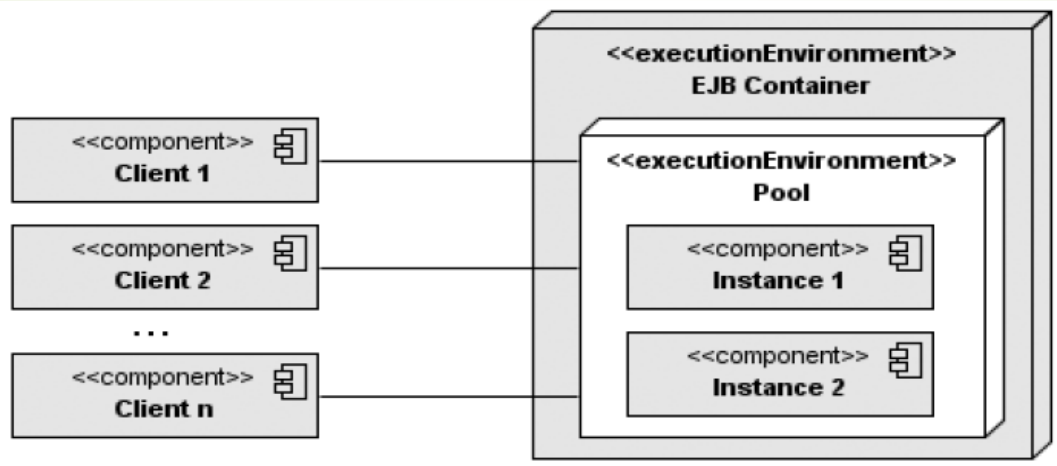


# *Stateless Beans*: Definição

- São os componentes *session bean* mais utilizados
- Mas o que *stateless* significa?
  - Significa que a tarefa tem que ser executada em uma única chamada de método
  - Ou seja, não há manutenção do estado entre chamadas de métodos do *session bean*
- Por isso, normalmente, possui um conjunto coeso de métodos relacionados
  - Exemplo: persistirLivro(), atualizarLivro(), buscarLivro(id)

# *Stateless Beans* e o Relacionamento com os Clientes

- *Stateless bean* é o tipo de *session bean* mais escalável, pois pode suportar um grande número de clientes
- Isso é alcançado por meio do uso de *pools* de *beans*
  - Ou seja, *stateless beans* são compartilhados entre diferentes clientes
  - O container EJB mantém um número limitado de *beans* na memória os quais podem ser compartilhados
  - Só é possível porque o estado não é mantido!



# *Stateful Beans*: Definição

- São componentes que mantêm o estado conversacional de uma interação

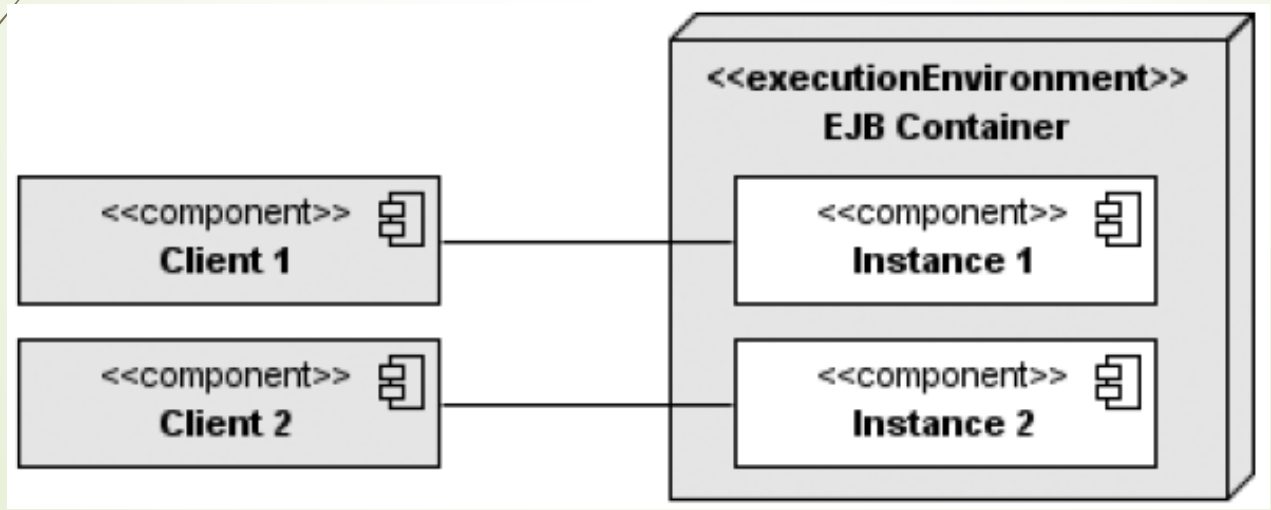
- Normalmente, são utilizados para tarefas que podem ser executadas em vários passos, onde cada um depende do anterior

# O Exemplo do Carrinho de Compras

```
Livro livro = new Livro();
livro.setTitulo("O Guia do Mochileiro das Galáxias");
livro.setPreco(38.5F);
livro.setDescricao("Humor e ficção científica");
livro.setNumPaginas(380);
carrinhoComprasEJB.adicionaItem(livro);
livro = new Livro ();
livro.setTitulo("O Restaurante no Fim do Universo");
livro.setPreco(34.5F);
livro.setDescricao("Humor e ficção científica");
livro.setNumPaginas(335);
carrinhoComprasEJB.adicionaItem(livro);
carrinhoComprasEJB.finalizaCompra();
```

# *Stateful Beans* e o Relacionamento com os Clientes

- Quando um cliente invoca um *stateful session bean*, o container EJB precisa prover a mesma instância para as chamadas subsequentes



# *Stateful Beans*: Questões de Desempenho

- O relacionamento um para um entre o cliente e o EJB pode custar caro!
  - Em uma aplicação com 1 milhão de clientes teríamos 1 milhão de EJBs instanciados
- Para minimizar essa sobrecarga em termos de memória consumida, *stateful beans* são retirados temporariamente da memória durante o período de inatividade
  - Esta técnica é conhecida como *passivation* e *activation*
- *Passivation* é o processo de remoção da memória e armazenamento
- *Activation* é o processo inverso

# *Stateful Beans*: Implementando o Carrinho de Compras como um EJB

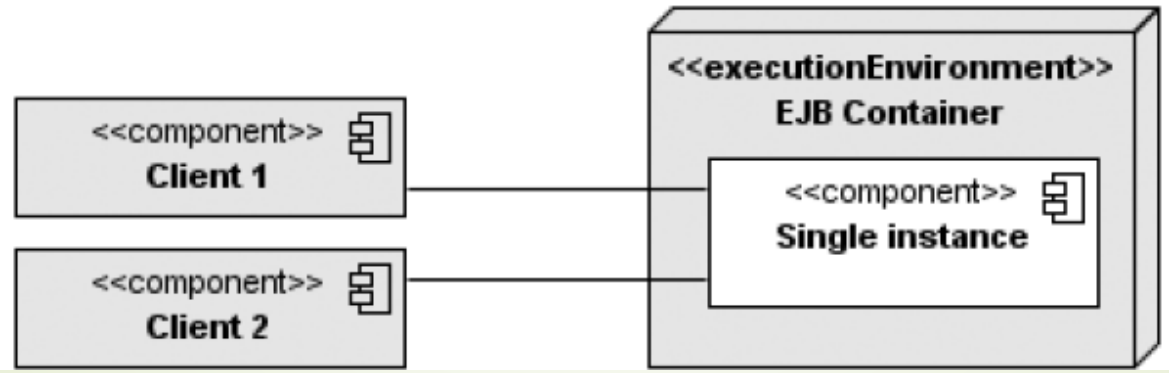
```
 9 @Stateful
10 @StatefulTimeout(20000)
11 public class CarrinhoComprasEJB {
12
13     private List<Item> itens = new ArrayList<Item>();
14
15     public void adicionaAoCarrinho(Item item) {
16         if (!itens.contains(item))
17             itens.add(item);
18     }
19
20     public void removeItem(Item item) {
21         if (itens.contains(item))
22             itens.remove(item);
23     }
24
25     public Float getTotal() {
26         Float total = 0f;
27         for (Item item : itens) {
28             total += (item.getPreco());
29         }
30         return total;
31     }
32
33     @Remove
34     public void checkout() {
35         // lógica de negócio ...
36         itens.clear();
37     }
38 }
```

- Duas anotações (opcionais) devem ser observadas: @StatefulTimeout e @Remove

- Ambas devem ser utilizadas para remover o EJB da memória
  - A primeira após um período de inatividade
  - E a segunda quando o método for chamado
- A idéia destas anotações é, em vez de confiar na remoção automática no momento em que a sessão do usuário terminar, remover no momento mais oportuno
  - Pode representar um menor consumo de memória

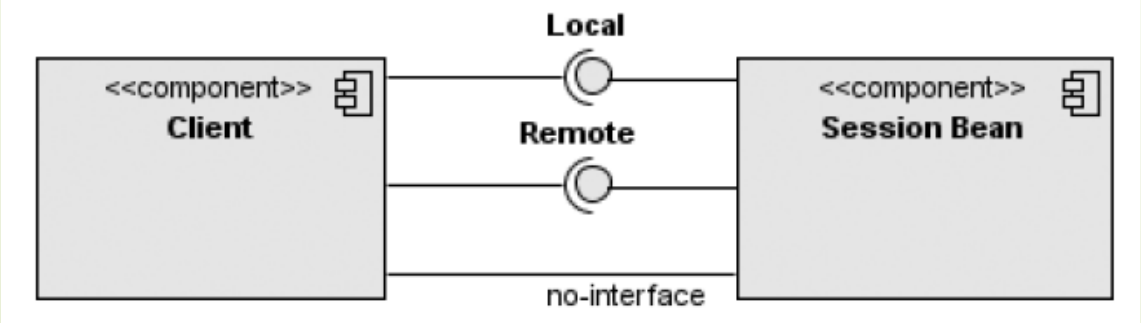
# *Singletons*

- Análogos ao padrão definido em Gamma *et al.* (1995)
- Logo, são úteis para permitir acesso à recursos e serviços compartilhados, quando é interessante manter o estado mas diversas instâncias não são necessárias
- Usa-se com a anotação @Singleton
  - O desenvolvedor não deve se preocupar em fazer o construtor da classe "private" nem criar o método estático getInstance(). Tudo é feito automaticamente



# As Formas de Invocação EJB

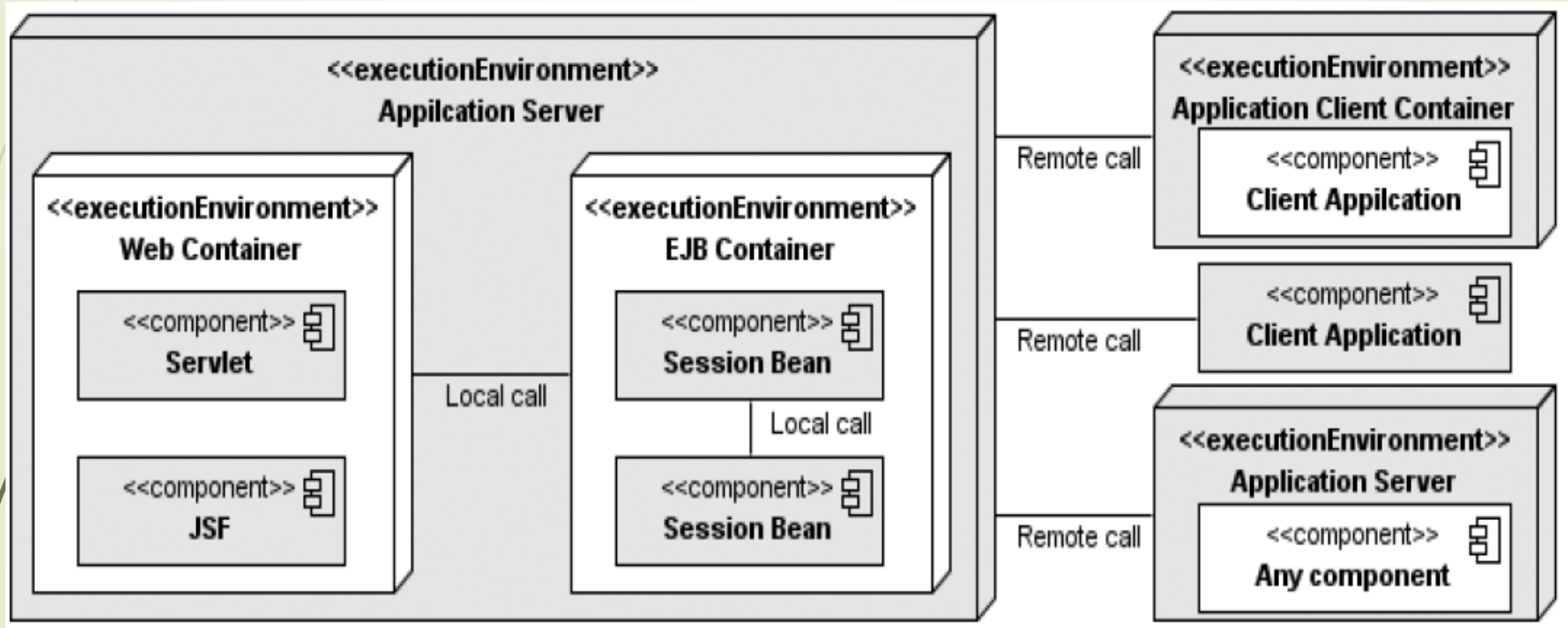
- Existem dois elementos principais no uso de EJBs:
  - *Business interfaces*: São interfaces que contém declarações dos métodos de negócio os quais são visíveis ao cliente e implementados por um *session bean*
  - *Session bean*: São *beans* que contém a implementação de métodos de negócio e podem possuir zero ou mais de uma interface (local e remota)
- Assim temos três formas de invocar um *session bean*



# As Formas de Invocação EJB

- *Remoto*: denota uma interface de negócio remota invocada através do protocolo RMI.
  - Parâmetros de métodos são passados por valor e precisam ser serializáveis
  - Usa-se a anotação @Remote (na interface)
- *Local*: denota uma interface de negócio local.
  - Parâmetros de métodos são passados por referência (como se fosse)
  - Usa-se a anotação @Local (na interface)
- *Sem interface:* nesse caso, o comportamento é análogo ao da forma *local*, mas não há definição de uma interface Java
- Observação: em grande parte das aplicações Web os EJBs residem no mesmo container. Logo, as formas *local* e *sem interface* podem ser usadas.

# As Formas de Invocação EJB



# Definindo Interfaces

- Existem duas formas de se definir interfaces *locais* e *remotas*
  - Uma anotação na própria interface

**Interfaces**

```
@Local
public interface LivroLocal {
   //...
}
@Remote
public interface LivroRemote{
   //...
}
```

**EJB**

```
@Stateless
public class LivroEJB implements LivroLocal, LivroRemote {
   // ...
}
```

  - Anotando o EJB que implementa a interface

**Interfaces**

```
public interface LivroLocal {
   //...
}
public interface LivroRemote{
   //...
}
```

**EJB**

```
@Stateless
@Local(LivroLocal)
@Remote(LivroRemote)
@LocalBean // para permitir a invocação sem interface
public class LivroEJB implements LivroLocal, LivroRemote {
   // ...
}
```

# A Visão do Cliente: Como Clientes invocam EJBs?

- Através de injeção de dependência usando anotações

```
@EJB private LivroRemote livroEJB;
@EJB private LivroLocal livroEJB;
@EJB private LivroEJB livroEJB;
```

- Ou usando JNDI
  - Permite executar serviços de um container EJB dentro de uma aplicação Java SE
    - Ótimo para testes de "unidade"

# O Ciclo de Vida dos EJBs: *Stateless* e *Singleton*

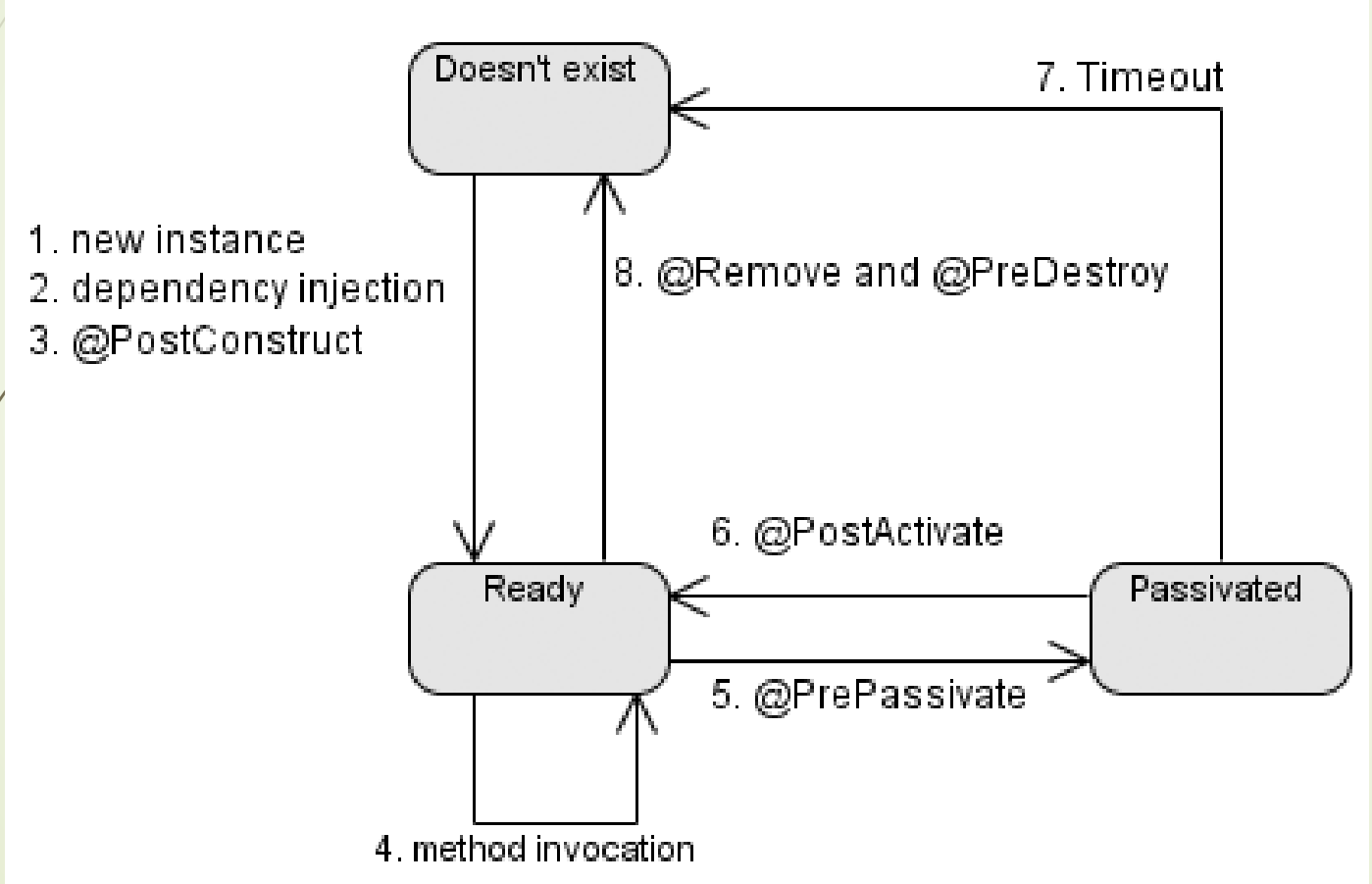
1) O ciclo de vida de um *stateless* ou *singleton bean* é iniciado quando um cliente requer uma referência ao *bean*
2) Se a nova instância usa injeção de dependência, o container as injeta
3) Se a instância tem um método anotado com @PostConstruct, o container invoca
4) O *bean* é instanciado e fica pronto para futuras chamadas (existe um *pool* de instâncias de *stateless*)
5) Quando o container não precisa mais da instância, o método com anotção @PreDestroy é chamado, se existir

# O Ciclo de Vida dos EJBs: *Stateless* e *Singleton*

# O Ciclo de Vida dos EJBs: *Stateful*

1) O ciclo de vida de um *stateful bean* é iniciado quando um cliente requer uma referência ao *bean*
2) Se a nova instância usa injeção de dependência, o container as injeta
3) Se a instância tem um método anotado com @PostConstruct, o container invoca
4) O *bean* executa a operação e é mantido em memória para requisições futuras
5) Se o cliente permanece inativo por um período chama o método anotado com @PrePassivate, se existir, e o *bean* passivo
6) Se o cliente invoca um *bean* apassivado o container o restaura para memória e chama o método anotado com @PostActivate, se existir
7) Se o cliente não invocar um *bean* apassivado durante um período determinado ele é destruído
8) Alternativamente ao passo 7, se o cliente tiver um método anotado com @Remove, o container invocará o método anotado com @PreDestroy, se existir

# O Ciclo de Vida dos EJBs: *Stateful*



# Agenda

- ✓ Arquiteturas Web em Java
- ✓ Arquitetura Java EE
- ✓ Introdução a Enterprise Java Beans (EJB)
- ➢ **Discussão: POO X Programação Orientada a Componentes (POC)**
- Quando usar EJBs?

# POO X POC

| Capabilities | SP | OOP | COP |
|---|---|---|---|
| *Divide and Conquer*<br>• Manage complexity<br>• Break a large problem down into smaller pieces | | | |
| *Unification of Data and Function*<br>• A software entity combines data and the functions processing those data<br>• Improves cohesion | X | | |
| *Encapsulation*<br>• The client of a software entity is insulated from how that software entity's data is stored or how its functions are implemented<br>• Reduces coupling | X | | |
| *Identity*<br>• Each software entity has a unique identity | X | | |
| *Interface*<br>• Represents specification dependency<br>• Divides a component specification into interfaces<br>• Restricts inter-component dependency | X | X | |
| *Deployment*<br>• The abstraction unit can be deployed independently | X | X | |

# Discussão: POO X Programação Orientada a Componentes (POC)

- POO X POC
  - POC é baseado em interfaces, POO é baseado em objetos;
  - POC é tecnologia para empacotar e distribuir, POO é tecnologia para implementar;
  - POC apoia reuso de alto nível, POO apoia reuso de baixo nível;
  - Componentes POC podem ser escritos em qualquer linguagem, POO é amarrada a linguagens orientadas a objeto;

# Discussão: POO X Programação Orientada a Componentes (POC)

- POO X POC (cont.)
  - POC visa baixo acoplamento, POO pode ter objetos fortemente dependentes entre si através de heranças;
  - POC geralmente faz uso de componentes de granularidade maior do que POO;
  - POC suporta múltiplas interfaces e projeto baseado em interfaces, POO não provê interfaces claras entre superclasses e subclasses;
  - POC apoia mais formas de associação dinâmica e descobrimento dinâmico, POO provê apoio limitado a recuperação de objetos e composição em tempo de execução.

# Discussão: POO X Programação Orientada a Componentes (POC)

- POC X POO (cont.)
  - POC tem melhores mecanismos para composição por terceiros, POO tem conectores limitados (invocação de métodos);
  - POC projeta componentes obedecendo a um modelo de componentes, POO projeta componentes obedecendo a princípios OO.

WANG, A.J.A., QIAN, K., “Component Oriented Programming”, Wiley Interscience, 2005. – Capítulos 1 e 2 (exceto CSL).

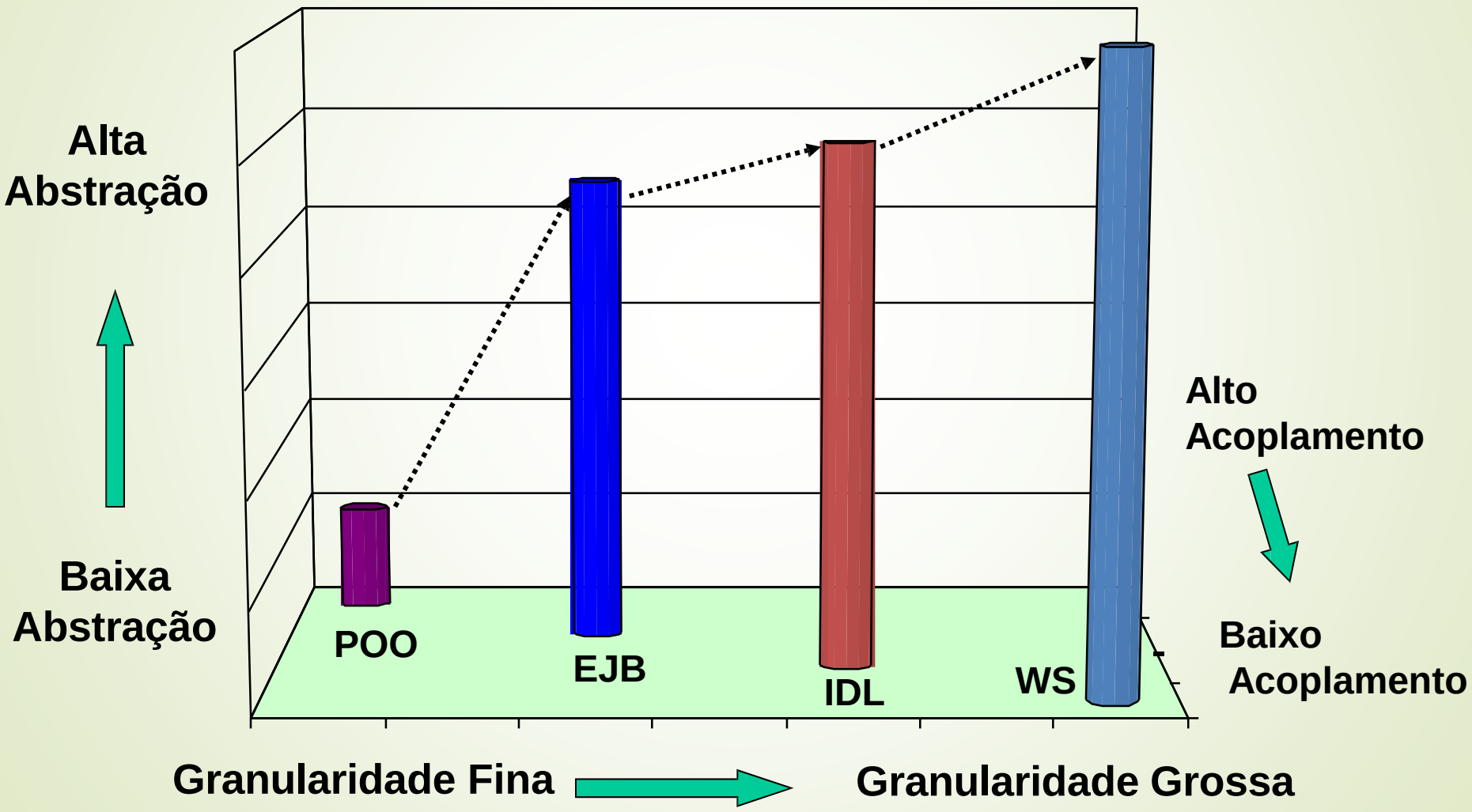
# Infra-Estrutura de Componentes (Complemento Teórico)

- *Framework* para facilitar a construção e a gerência de componentes. Consiste de três modelos:
  - Modelo de Componentes, ele define:
    - O que é um componente válido;
    - Como criar um novo componente na infra-estrutura;
  - Modelo de Conexão, ele define:
    - Coleção de conectores e facilidades para agrupar componentes.
  - Modelo de Implantação (*Deployment*), ele define:
    - Como colocar componentes em um ambiente para que possam ser utilizados.

# Modelos de Componentes (Complemento Teórico)

- Para que a arquitetura de implementação possa ser especificada de maneira adequada, determinadas regras e condições precisam ser respeitadas, de forma que os componentes consigam interagir no contexto da tecnologia utilizada.
  - Como as interfaces são especificadas?
  - Como as interfaces são nomeadas?
  - Existem interfaces obrigatórias na implementação dos componentes?
- modelos de componentes como da especificação EJB, CORBA e de Serviços Web definem como seus componentes devem ser descritos.

# POC vs POO


Alta
Abstração
Baixa
Abstração
POO
EJB
IDL
WS
Alto
Acoplamento
Baixo
Acoplamento
Granularidade Fina
Granularidade Grossa

# Agenda

- ✓Arquiteturas Web em Java
- ✓Arquitetura Java EE
- ✓Introdução a Enterprise Java Beans (EJB)
- ✓Discussão: POO X Programação Orientada a Componentes (POC)
- ➢**Quando usar EJBs?**

# Quando usar EJBs?

- Sua aplicação necessita “escalar” para um número grande de usuários.
  - Ciclo de vida gerenciado pelo container
    - Instancia um número apropriado de EJBs para atender à demanda dos clientes ativos.
  - Distribuição dos EJBs por diversas JVMs e Servidores.
    - Permite distribuição das tarefas entre as JVMs e Servidores de forma transparente (balanceamento de carga realizado pelo container).
  - Possibilidade do uso de MDBs para balanceamento de carga entre diversos servidores.

# Quando usar EJBs?

- Sua aplicação gerencia dados relativamente complexos que necessitam suporte transacional.
  - O container realiza a gerência transacional.
  - JTA permite demarcar onde diversas operações de EJB devem ser tratadas como atômicas.

# Quando usar EJBs?

- Sua aplicação tem possui diversos e diferentes tipos de clientes.
  - Há diversos tipos de clientes que podem invocar EJBs.
    - Aplicações Cliente Java SE
    - Outros componentes EJB
    - Componentes Web Java EE
    - Clientes de Web Services

# Agenda

- ✓Arquiteturas Web em Java
- ✓Arquitetura Java EE
- ✓Introdução a Enterprise Java Beans (EJB)
- ✓Discussão: POO X Programação Orientada a Componentes (POC)
- ✓Quando usar EJBs?

# Leituras Sugeridas

- “Java EE 7 Tutorial”, Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito Parte “Introduction to EJB”
- “Java EE 7: The Big Picture”, Danny Coward Capítulo 9

# Arquitetura JEE
# Introdução à Camada de Negócios: Enterprise Java Beans (EJB)