J530 - Enterprise JavaBeans

Introdução a EJB e Stateless Session Beans

Helder da Rocha (helder@acm.org)

Componentes de um EJB

- Para que o container possa gerar o código necessário é preciso que o bean seja implantado corretamente. Um EJB deve estar empacotado em um JAR contendo
  - Uma classe que implementa o bean
  - Interface(s) do(s) interceptador(es) (Remote ou Local)
  - Interface(s) de fábrica: Home (Remote ou Local)
  - Deployment descriptor
- A estrutura é definida na especificação e formalizada em interfaces, esquemas XML e validadores
  - Antes de implantar uma aplicação, o servidor verifica se todos os itens estão de acordo com a especificação. Se um não estiver, a aplicação poderá não ser instalada

Classe EJB

- É o objeto distribuído. Contém a lógica de negócio
  - Pode também implementar rotinas de persistência
- É uma classe Java que implementa os mesmos métodos declarados nas interfaces do interceptador
  - Não é objeto remoto (não implementa java.rmi.Remote)
- Implementações diferem, dependendo do tipo
  - Session Beans - lógica relacionada a processos de negócio (computar preços, transferir fundos)
  - Entity Beans - lógica relacionada a dados (mudar o nome de um cliente, reduzir o saldo)
  - Message-driven Beans - lógica orientada a eventos (lógica assíncrona)

Interfaces padrão do EJB

- Todo bean implementa a *javax.ejb.EnterpriseBean*

```
package javax.ejb;
public interface EnterpriseBean {
}
```

- Na verdade, todo bean implementa uma interface derivada de EnterpriseBean

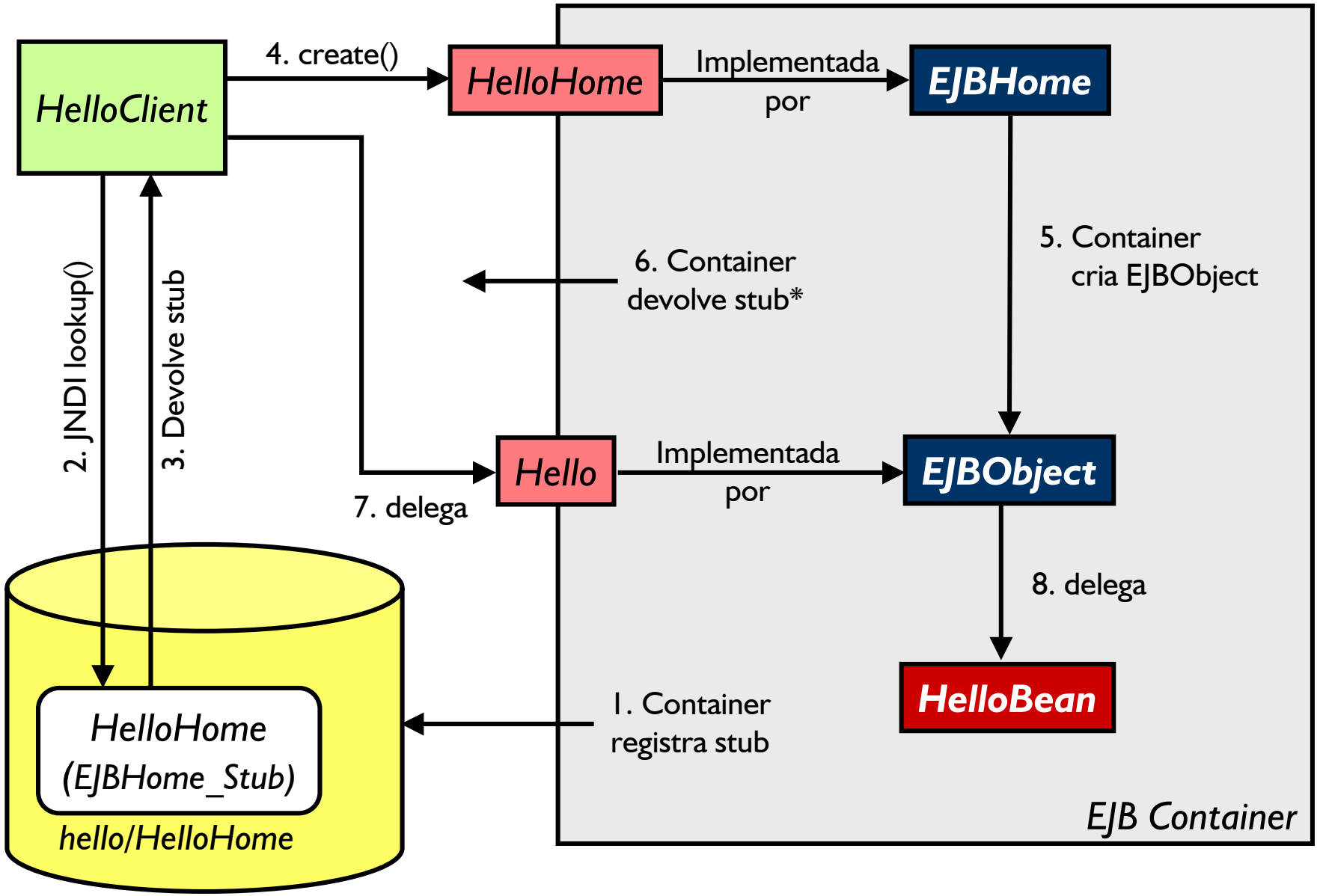
```
SessionBean extends EnterpriseBean
EntityBean extends EnterpriseBean
MessageDrivenBean extends EnterpriseBean
```

- Cada interface tem métodos próprios que precisam ser implementados em cada bean
  - Além deles, a especificação pode exigir outros

Interface do Componente

- *Component interface*. *Há dois tipos*
  - *Remote component interface*
  - *Local component interface*
- *Para criar uma interface de componente, é preciso estender interfaces EJBObject ou EJBLocalObject*
  - *O container criará automaticamente interceptadores locais ou remotos contendo todos os métodos declarados*
  - *Um bean só precisa de um tipo de interceptador (ou Local ou Remoto) mas pode ter ambos, se necessário*
- *Interceptador EJBObject é objeto remoto RMI-IIOP*
  - *É gerado pelo container e delega chamadas ao bean*
  - *Interface javax.ejb.EJBObject estende java.rmi.Remote*
  - *Define métodos de negócio remotos expostos pelo bean*

Arquitetura (usando acesso remoto apenas)

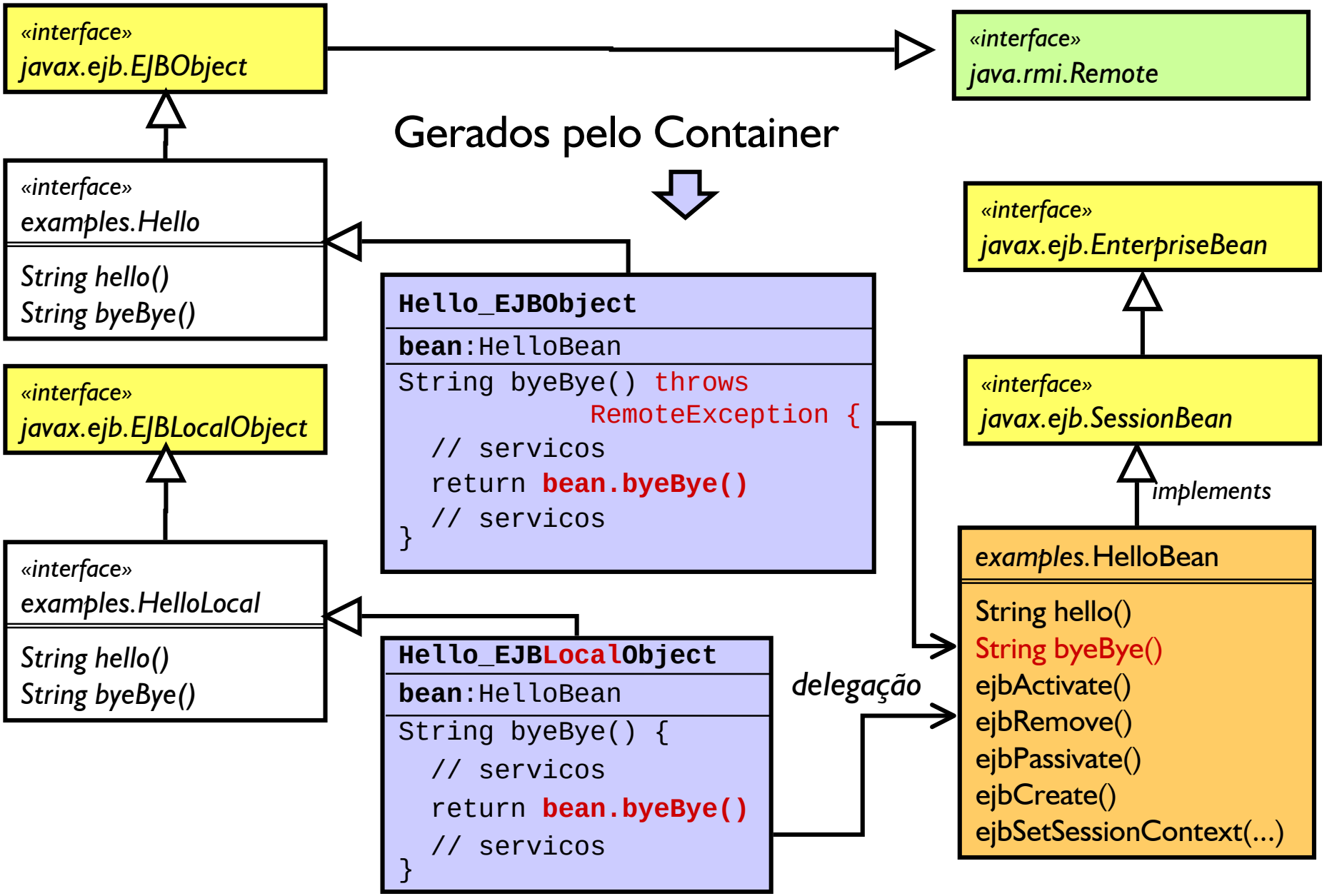


* *detalhes como comunicacão entre stubs e skeletons foram omitidos*

EJB Object

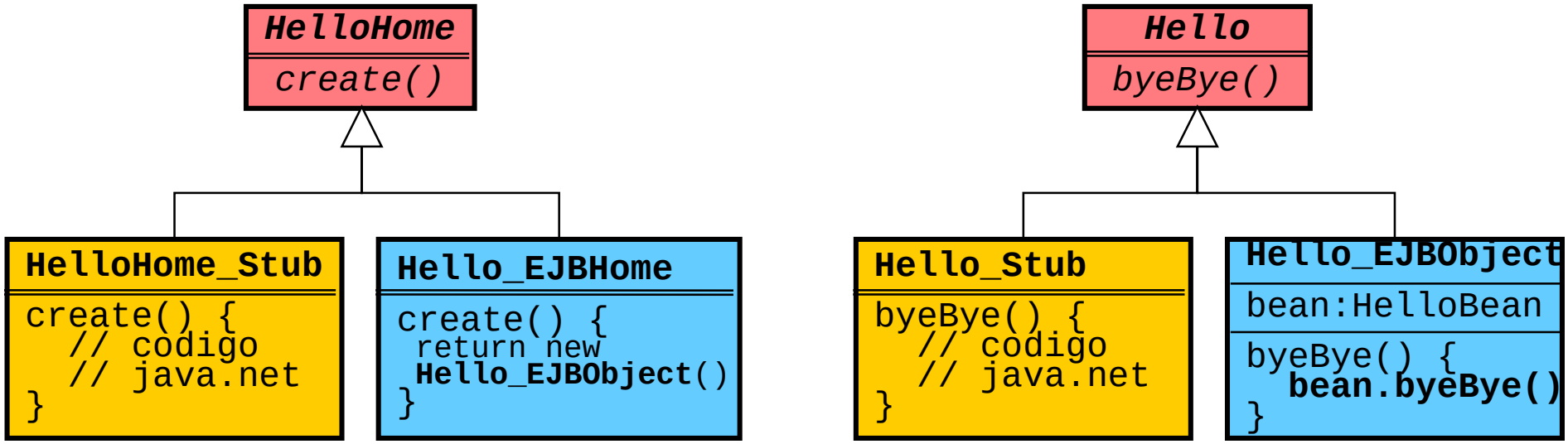
- Enterprise Beans podem ser acessados remotamente, mas não são objetos remotos
  - Um cliente (ou o skeleton) nunca chama o método diretamente em uma instância do bean: a chamada é interceptada pelo Container e delegada à instância
  - Ao interceptar requisições, o container pode automaticamente realizar middleware implicito
- O **EJBObject** é quem implementa a interface remota. Ele é o interceptador que chama o Bean
  - É objeto inteligente que sabe realizar a lógica intermediária que EJB container requer antes que uma chamada seja processada
  - Expõe cada método de negócio do bean e delega as requisições
- O container gera o EJBObject a partir da interface criada pelo programador para o componente

EJBObject, EJBLocalObject e Enterprise Bean



Objetos gerados pelo container

- *Durante o deployment*
  - *Interceptadores remotos EJBHome e EJBObject são gerados*
  - *Interceptadores locais EJBLocalHome e EJBLocalObject são gerados*
  - *Stubs são gerados para os interceptadores remotos. Stub da interface Home é mapeado a nome no servidor JNDI*
- *Hierarquias dos objetos gerados pelo container**



- *EJBObject delega requisições para o Bean*

** Apenas para efeito de ilustração. Implementação exata depende, na verdade, do fabricante*

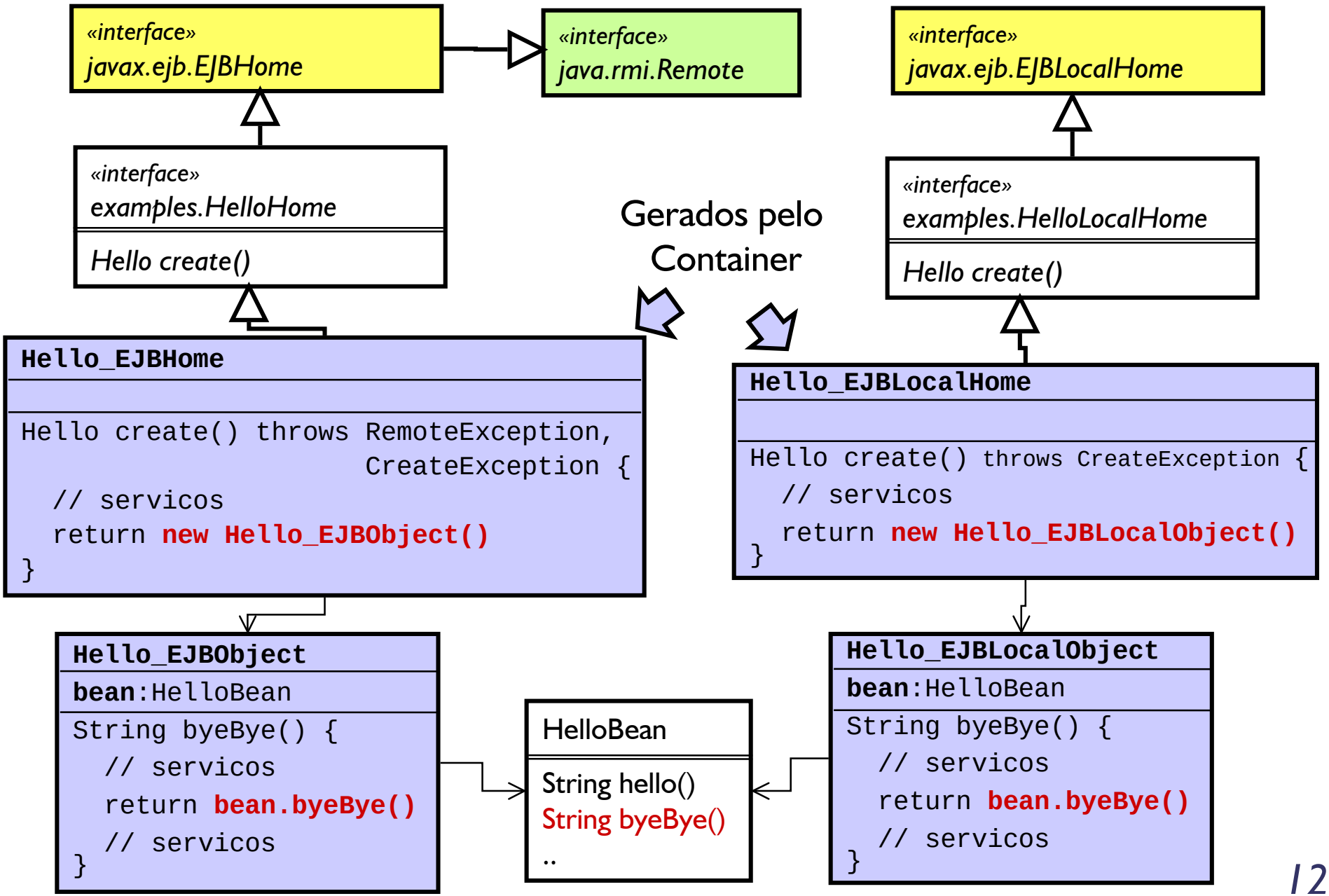
RMI-IIOP versus EJB Objects

- *Qualquer objeto que implementa javax.rmi.Remote é um objeto remoto e chamável de outra JVM*
  - *Todos os EJB Objects (gerados pelo Container) são objetos RMI-IIOP*
  - *Interfaces Remote EJB são interfaces Remote RMI-IIOP, só que precisam aderir à especificação EJB (têm métodos adicionais que o container implementa)*
- *Regras de java.rmi.Remote valem para EJBObjects*
  1. *Todos os métodos provocam RemoteException*
  2. *Interfaces Remote precisam suportar tipos de dados que podem ser passados via RMI-IIOP.*
     Tipos permitidos: primitivos, objetos serializáveis e objetos remotos RMI-IIOP

EJB Home

- Fábrica usada para que clientes possam adquirir referências remotas a objetos EJB
  - Evita ter que registrar cada objeto: registra-se só a fábrica
  - Para obter uma referência a um objeto EJB, cliente chama métodos do EJBHome
  - O EJBHome é responsável por operações do ciclo de vida de um objeto EJB (criar, destruir, encontrar)
- Objetos **EJBHome** servem para
  - Criar objetos EJB
  - Encontrar objetos EJB existentes (em Entity Beans)
  - Remover objetos EJB
- São parte do container e gerados automaticamente
  - Implementam interface Home definida pelo programador

EJB(Local)Home e EJB(Local)Objects



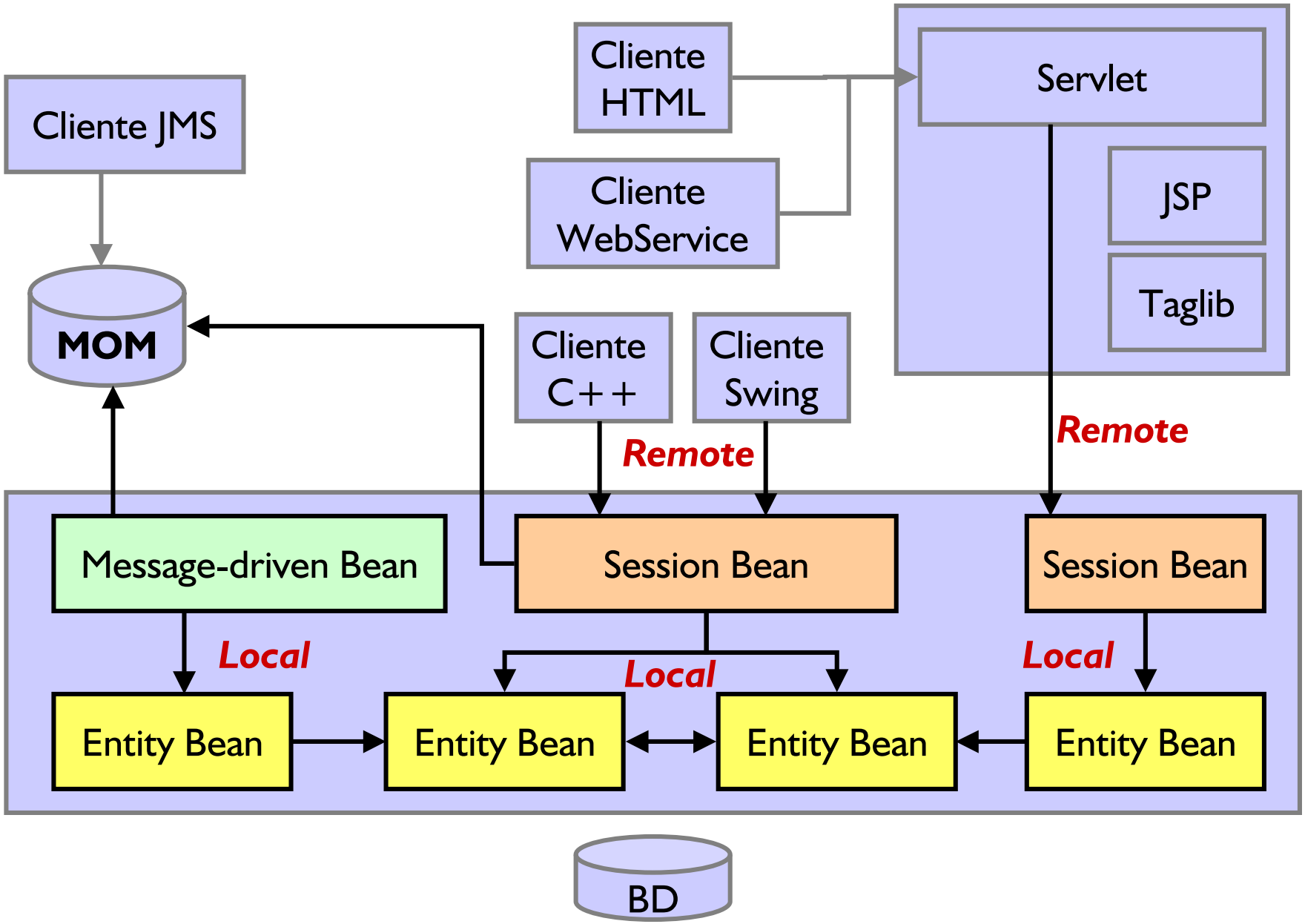
Por que usar interfaces locais

- Um problema das interfaces remotas é que criar objetos através dela é um processo lento
  1. Cliente chama um stub local
  2. Stub transforma os parâmetros em formato adequado à rede
  3. Stub passa os dados pela rede
  4. Esqueleto reconstrói os parâmetros
  5. Esqueleto chama o EJBObject
  6. EJBObject realiza operações de middleware como connection pooling, transações, segurança e serviços de ciclo de vida
  7. Depois que o EJBObject chama o Enterprise Bean, processo é repetido no sentido contrário
- Muito overhead! Em EJB 2.0 é possível chamar EJBs através de sua interface local
  1. Cliente chama objeto local
  2. Objeto EJB local realiza middleware
  3. Depois que o trabalho termina devolve o controle a quem chamou

Interfaces locais: conseqüências

- Interfaces locais são definidas para objetos EJBHome e para EJBObject (interfaces EJBLocal e EJBLocalHome)
- Benefícios
  - Pode-se escrever beans menores para realizar tarefas simples sem medo de problemas de performance
  - Uso típico: fachadas Session que acessam Entities que não são acessíveis remotamente
- São opcionais
  - Substituem ou complementam as interfaces remotas existentes
- Efeitos colaterais
  - Só funcionam ao chamar beans do mesmo processo (não dá para mudar o modelo de deployment sem alterar o código)
  - Parâmetros são passados por referência e não por valor: muda a semântica da aplicação!!!
- Uso típico: Entity Beans geralmente só têm interfaces locais

Uso de interfaces locais e remotas

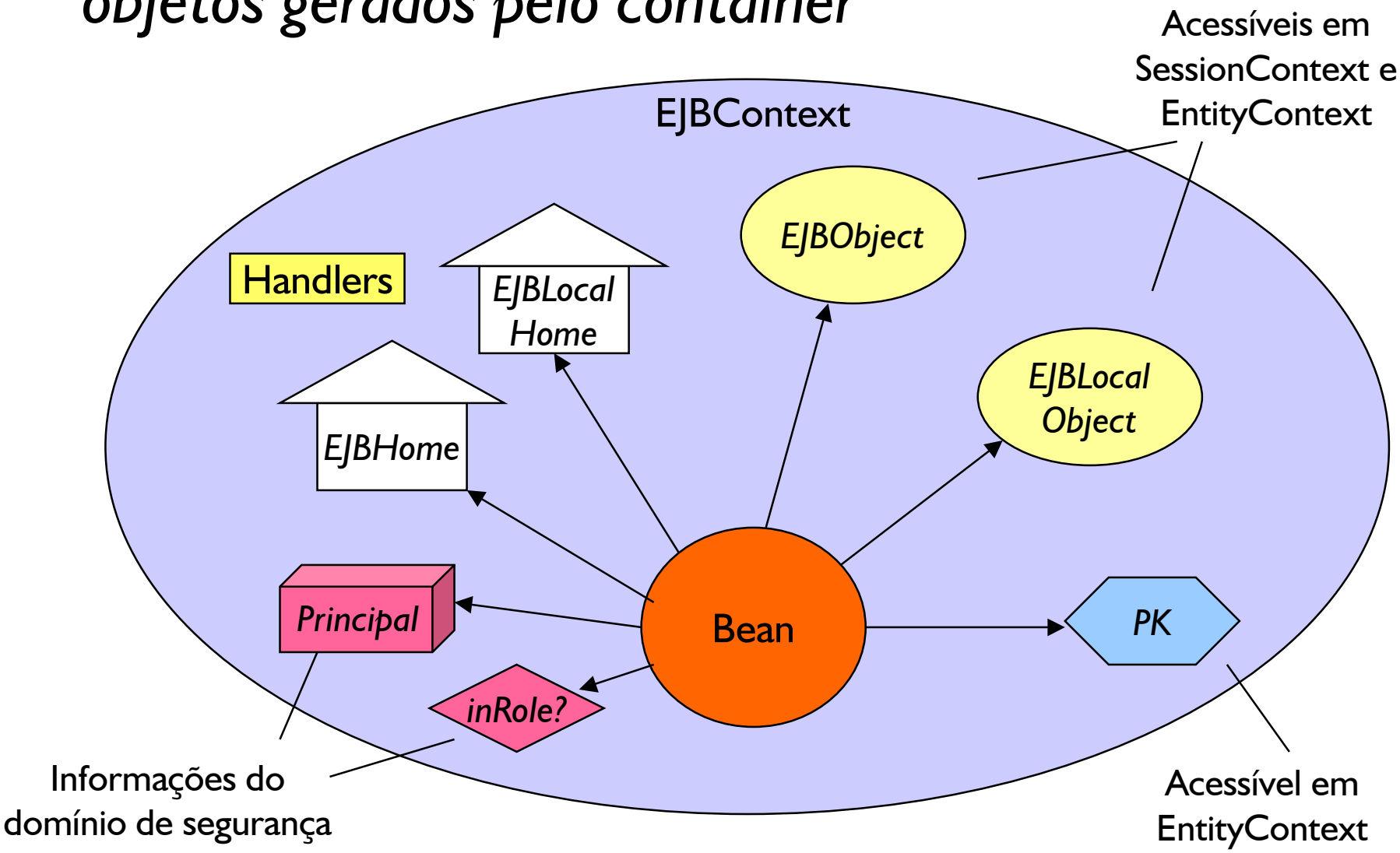


javax.ejb.EJBContext

- Usada em todos os beans como forma de ter acesso ao contexto de tempo de execução do bean
  - Todo bean tem um método get/set<Tipo>Context, onde <Tipo> é Entity, Session ou MessageDriven
- Através desse contexto, o bean pode ter acesso...
  - Ao objeto que implementa sua interface Home
  - Ao usuário e perfil do usuário logado
  - Ao status de transações
- Sub-interfaces de EJBContext acessam
  - O interceptador que implementa sua interface do componente (SessionContext e EntityContext)
  - A chave primária do Entity Bean

EJBContext

- *O contexto possui métodos para dar acesso aos objetos gerados pelo container*



Métodos de EJBContext

- Principais métodos definidos na Interface
  - javax.security.Principal getCallerPrincipal()
  - boolean isCallerInRole(String role)
  - EJBHome getEJBHome()
  - EJBLocalHome getEJBLocalHome()
- Métodos adicionais em SessionContext e EntityContext (sub-interfaces de EJBContext)
  - EJBObject getEJBObject()
  - EJBLocalObject getEJBLocalObject()
- Métodos adicionais em EntityContext
  - Object getPrimaryKey()

Deployment descriptors

- Para informar ao container sobre suas necessidades de middleware, o bean provider deve declarar suas necessidades de middleware em um deployment descriptor (arquivo der configuração em XML)
  - Informações de gerência e ciclo de vida (quem é o Home, o Remote, se é Session, Entity ou MDB)
  - Requisitos de persistencia para Entity Beans
  - Configuração de transações
  - Segurança (quem pode fazer o que com que beans, que metodos pode usar, etc.)
  - ...
- Pode-se usar uma ferramenta (geralmente a ferramenta do próprio container) para gerar o arquivo

Deployment descriptor simples: exemplo

```
<!DOCTYPE ejb-jar PUBLIC
"-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
 <enterprise-beans>
  <session>
   <ejb-name>Hello</ejb-name>
   <home>examples.HelloHome</home>
   <remote>examples.Hello</remote>
   <local-home>examples.HelloLocalHome</local-home>
   <local>examples.HelloLocal</local>
   <ejb-class>examples.HelloBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
  </session>
 </enterprise-beans>
 <assembly-descriptor>
   <!-- Configuração de serviços -->
   <!-- Autorização -->
   <!-- Transações -->
 </assembly-descriptor>
</ejb-jar>
```

ejb-jar.xml

Arquivos proprietários do fabricante

- Cada servidor tem recursos adicionais que podem ser definidos em arquivos específicos
  - Como configurar load-balancing
  - Clustering
  - Pooling
  - Monitoramento
  - Mapeamento JNDI
- Geralmente são gerados por ferramentas no próprio servidor
- Podem também ser codificados à mão (jboss.xml)
- Devem ser incluidas no bean antes do deployment

Arquivos proprietários do fabricante

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>ProdutoEJB</ejb-name>
         <table-name>produtos</table-name>
         <cmp-field>
            <field-name>name</field-name>
            <column-name>nome</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>codigo</field-name>
            <column-name>id</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>quantidade</field-name>
            <column-name>qte</column-name>
         </cmp-field>
                ...
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Mapeamentos para bancos de dados

jbosscmp-jdbc.xml

Mapeamentos para serviço de nomes

```
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>Hello</ejb-name>
      <jndi-name>hello/HelloHome</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```

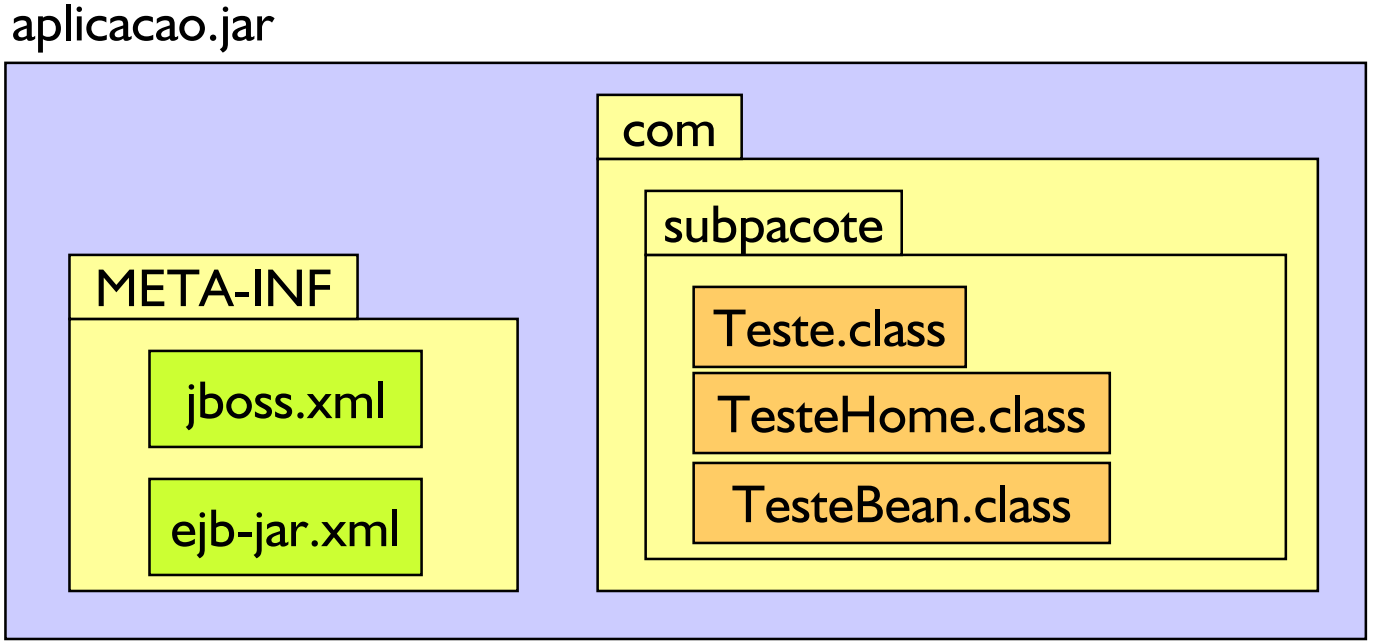
jboss.xml

EJB-JAR

- Arquivo JAR que contém tudo o que descrevemos
- Pode ser gerado de qualquer forma
  - ZIP, Ant, ferramentas dos containers
  - IDEs (NetBeans e deploytool)
- Uma vez feito o EJB-JAR, seu bean está pronto e é *unidade implantável* em application server
  - Ferramentas dos containers podem decomprimir, ler e extrair informações contidas no EJB-JAR
  - Depois, ferramentas realizam tarefas específicas do fabricante para gerar EJB Objects, EJB Home, importar seu bean no container, etc.
- Pode-se ter vários beans em um ejb-jar

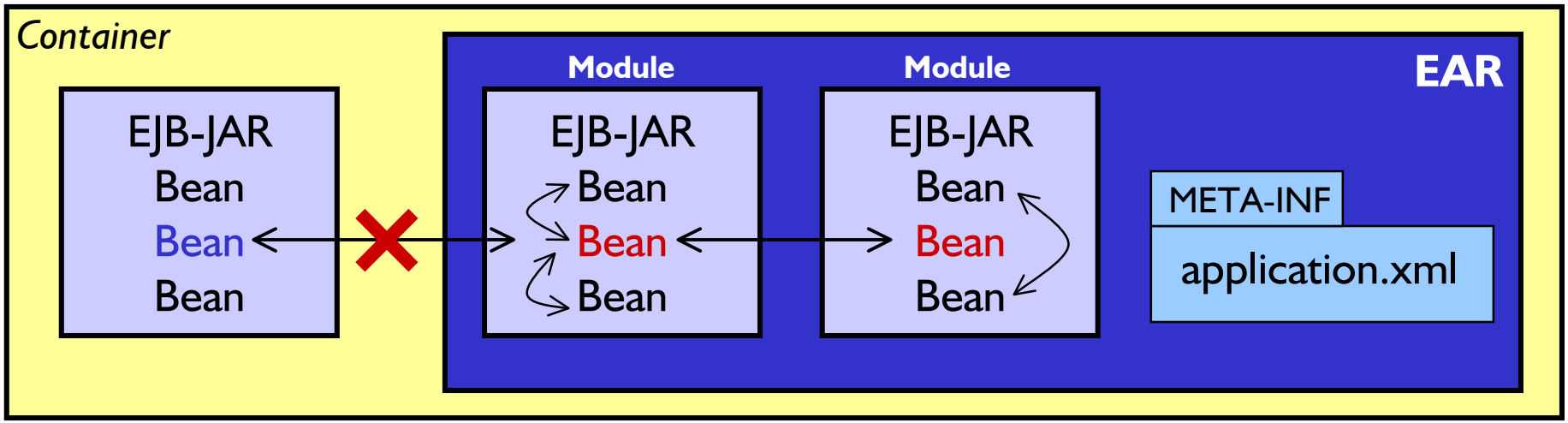
Estrutura de um EJB-JAR

- *O EJB-JAR é um JAR comum. Não possui uma estrutura especial exceto quanto à localização dos deployment descriptors (padrão e do fabricante).*
  - *Coloque as classes em seus pacotes a partir da raiz*
  - *Coloque os deployment descriptors no META-INF*



EARs

- *Classes em um componente JAR não são visíveis em outro componente JAR no container*
- *Para que beans em um JAR possam ter acesso a classes de um bean em outro componente, é preciso que esteja em um EAR*
  - *Há outras formas de resolver este problema, mas são todas não portáveis*



application.xml de um Enterprise ARchive

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC
'-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN'
'http://java.sun.com/dtd/application_1_3.dtd'>

<application>
  <display-name>ConverterEAR</display-name>
  <module>
    <ejb>Converter-ejb.jar</ejb>
  </module>
  <module>
    <java>utils.jar</java>
  </module>
  <module>
    <web>
      <web-uri>Converter-web.war</web-uri>
      <context-root>converter</context-root>
    </web>
  </module>
</application>
```

JARs conseguem ver classes uns dos outros

WAR vê classes dos JARs

JARs não vêem classes do WAR

Session Beans

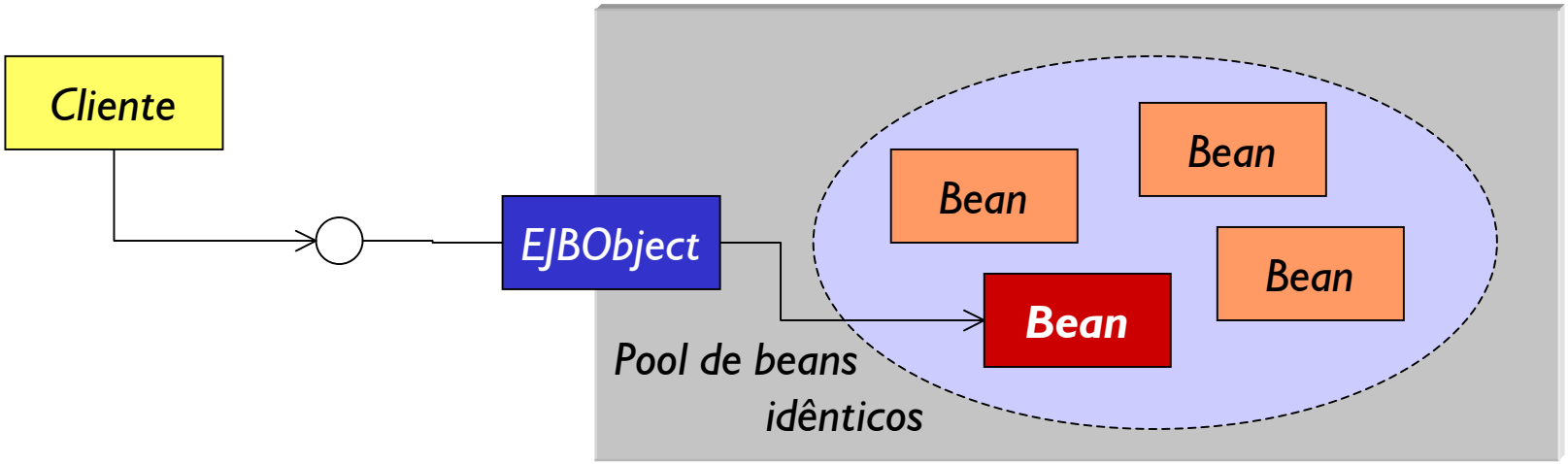
- São objetos de processo de negócio
  - Implementam lógica de negócio, algoritmos, workflow
  - Representam ações
- Uma das principais diferenças entre Session Beans e Entity Beans é o seu escopo de vida
  - Um Session Bean dura no máximo uma sessão (do cliente)
- Sessão
  - Tempo que o browser está aberto
  - Tempo que um outro bean usa o session bean
  - Tempo que uma aplicação remota está aberta
- Objetos transientes
  - Não tem seu estado armazenado em meio persistente

Tipos de Session Beans

- Clientes travam um diálogo com um bean (o diálogo é a interação entre um cliente e um bean)
  - Consiste de uma ou mais chamadas entre cliente e bean
  - Dura um processo de negócios para o cliente
- Os dois tipos de session beans modelam tipos diferentes de diálogos
  - **Stateful** Session Beans modelam diálogos consistem de várias requisições onde certas requisições podem depender do estado de requisições anteriores
  - **Stateless** Session Beans modelam diálogos que consistem de apenas uma requisição

Stateless Session Beans

- *Como stateless session beans não mantém informação de estado do diálogo, todas as instâncias do mesmo bean são equivalentes e indistiguíveis*
  - *Não importa que chamou o bean no passado*
  - *Qualquer instância disponível de um session bean pode servir a qualquer cliente*
- *Session Beans podem ser guardados em um pool, reutilizados e passados de um cliente para outro em cada chamada*



Fonte: [1]

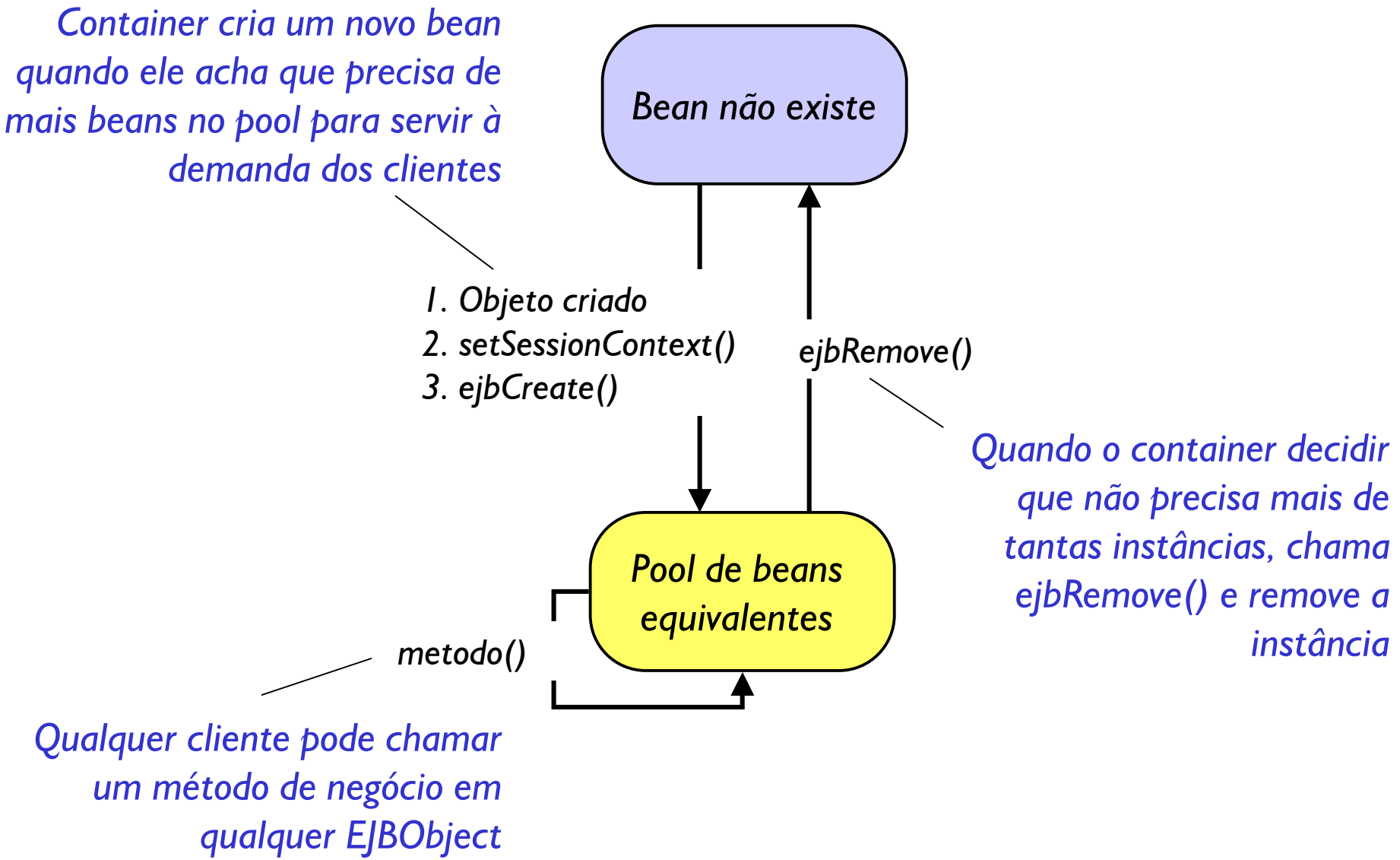
Métodos de um Stateless Session Bean

- *Por implementar a interface javax.ejb.SessionBean, cada Session bean precisa implementar os seguintes métodos*
- **`void setSessionContext(SessionContext ctx)`**
  - *Associa bean com contexto da sessão*
  - *O contexto pode ser usado para obter referências para o interceptador e home do bean, se necessário*
  - *Guarde a referência em uma variável de instância*
- **`void ejbCreate()`**
  - *Realiza a inicialização do bean. Pode ser vazio.*
- **`void ejbRemove()`**
  - *Chamado antes de liberar recursos e remover o bean da memória.*
  - *Pode ser vazio.*
- **`void ejbPassivate()`**
  - *Não utilizado por Stateless Session Beans. Deixe vazio.*
- **`void ejbActivate()`**
  - *Não utilizado por Stateless Session Beans. Deixe vazio.*

javax.ejb.SessionContext

- Usado para obter o contexto de tempo de execução do Session Bean
- Estende javax.ejb.*EJBContext* com dois métodos
  - `getEJBLocalObject()`: retorna referência para objeto interceptador local (gerado pelo container)
  - `getEJBObject()`: retorna referência para interceptador remoto (que é objeto Remote)
- Estes métodos podem ser usados quando o bean desejar passar uma instância de seu objeto remoto para algum método
  - Exemplo: situações onde a aplicação, se fosse local, usaria "this" para passar uma referência do objeto para outro através de um método. "this" não referencia o objeto remoto, mas o bean. A interface remota, porém, é implementada pelo objeto remoto

Ciclo de vida: Stateless Session Bean



Fonte: [1]

Exemplo: interfaces Remote e Home

```
package loja;
import java.rmi.RemoteException;
public interface Loja extends javax.ejb.EJBObject {
  public Collection listarProdutos() throws RemoteException;
  ...
}
```

```
package loja;
import java.rmi.RemoteException;
public interface LojaHome extends javax.ejb.EJBHome {
  Loja create()
      throws java.rmi.RemoteException, CreateException;
}
```

Compare create() de EJBHome com ejbCreate() do EnterpriseBean

Enterprise JavaBean

```
package loja;
public class LojaBean implements javax.ejb.SessionBean {
    private SessionContext sessionContext;

    public Collection listarProdutos() {
       System.out.println("listarProdutos() chamado");
       // implementação;
    }
    public void ejbCreate() throws CreateException {
       System.out.println("ejbCreate(val) chamado");
    }
    public void ejbRemove() {
       System.out.println("ejbRemove() chamado");
    }
    public void ejbActivate() {
       System.out.println("ejbActivate() chamado");
    }
    public void ejbPassivate() {
       System.out.println("ejbPassivate() chamado");
    }
    public void setSessionContext(SessionContext ctx) {
        this.sessionContext = ctx;
    }
}
```

Contrato

Deployment Descriptor

```
<!DOCTYPE ejb-jar PUBLIC
 "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
 "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>LojaEJB</ejb-name>
      <home>loja.LojaHome</home>
      <remote>loja.Loja</remote>
      <ejb-class>loja.LojaBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Conclusões

- *Stateless session beans são os beans RMI-IIOP (se remotos) mais simples*
  - *Permitem implementar qualquer aplicação distribuída que antes era implementada em RMI*
  - *Diferentemente do RMI, seu ciclo de vida é controlado pelo container, e podem usar serviços de persistência (DataSource), autenticação, autorização e transações fornecidos pelo container*
- *Não defina atributos de instância em Stateless Session Beans!*
  - *Eles podem ser compartilhados por outros clientes ou até desaparecer, pois o container não é obrigado a manter stateless session beans ativos.*

Exercicio opcional*

- *1. Implemente um session bean com a seguinte interface*

```
public interface DataSession {
   public String dataHoje();
   public int soma(int a, int b);
}
```

a) Transforme a interface acima em uma interface de componente Remota

b) Crie uma interface Home e classe EnterpriseBean

d) Crie e preencha um deployment descriptor

e) Crie o jboss.xml e empacote tudo

f) Escreva um cliente que chame os métodos do bean

** Stateless Beans foram demonstrados e criados no capítulo 1. Faça este exercício se nenhum similar foi feito no capítulo 1*

Fontes

[1] *Ed Roman, Mastering EJB 2, 2002, Capítulo 4.*

[2] *Dale Green. Session Beans. J2EE Tutorial, Sun*

[3] *Linda de Michiel et al. Enterprise JavaBeans 2.1 Specification. Sun Microsystems, 2003.*

Curso J530: Enterprise JavaBeans

Revisão 2.0 - Junho de 2003

(helder@acm.org)

argonavis.com.br