Java 2 Enterprise Edition

Helder da Rocha
www.argonavis.com.br

Session Beans

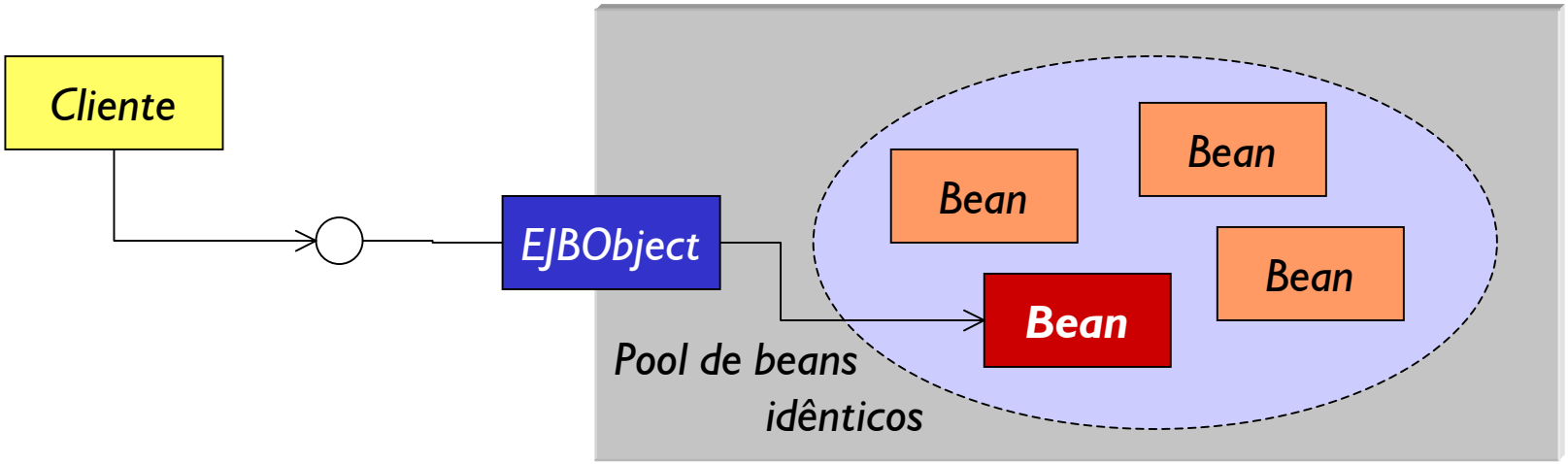
- São objetos de processo de negócio
  - Implementam lógica de negócio, algoritmos, workflow
  - Representam ações
- Uma das principais diferenças entre Session Beans e Entity Beans é o seu escopo de vida
  - Um Session Bean dura no máximo uma sessão (do cliente)
- Sessão
  - Tempo que o browser está 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 quem 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]

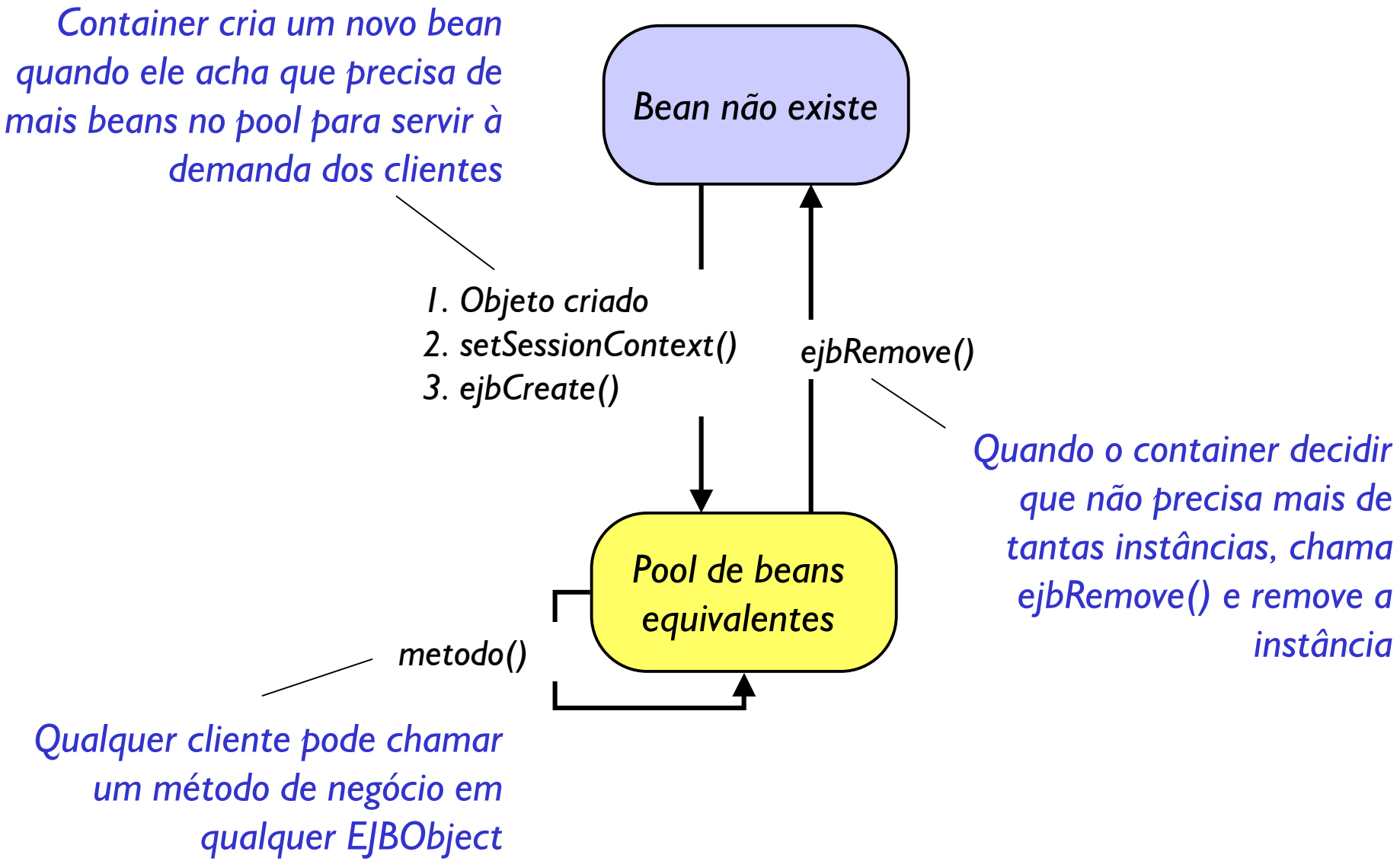
Métodos de um Stateless Session Bean

- *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>
```

Resumo

- *Stateless session beans são os beans RMI-IIOP 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

- *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

c) Crie a classe EnterpriseBean

d) Crie e preencha um deployment descriptor

e) Crie o jboss.xml e empacote tudo

f) Escreva um cliente que acesse o bean e chame seus métodos

Exercício extra

- *2. Escreva um Stateless Session Bean chamado Loja contendo a seguinte interface*

```
public Collection listarProdutos();
public void criarProduto(String cod, String nome,
                         BigDecimal preco, int qte);
public void removerProduto(String cod);
public ProdutoVO mostrarProduto(String cod);
```

  - *A Collection é uma ArrayList de loja.ProdutoVO.*
  - *Os métodos devem acessar o banco de dados e a tabela de produtos (criada em exercícios anteriores). Chame os métodos do ProdutoDAO (fornecido) que já contém código para fazer isto.*
  - *Uma vez feito o deployment sem erros, escreva um cliente e execute-o (o classpath do cliente deve conter a classe ProdutoVO)*
  - *Use como guia, se desejar, exemplos do capítulo 1 para construir jboss.xml e cliente.*

Stateful Session Beans

- Quando um cliente chama um método de um bean, ele está iniciando um diálogo
  - O estado do diálogo deve ser mantido para a próxima requisição
  - Container não pode fazer o mesmo tipo de pooling que faz com stateless session beans
- Passivação e ativação
  - Solução para o problema de pooling
  - Dados do bean são armazenados em meio persistente durante a passivação e recuperados na ativação
  - Permite manter poucas instâncias no ar e vários clientes
  - Estratégia comum: LRU - Least Recently Used - Se container precisar de recursos, beans menos usados serão passivados. Logo que receberem uma requisição, serão reativados
- Objetos são serializados, portanto, fazem parte do estado do diálogo apenas objetos e variáveis não transientes

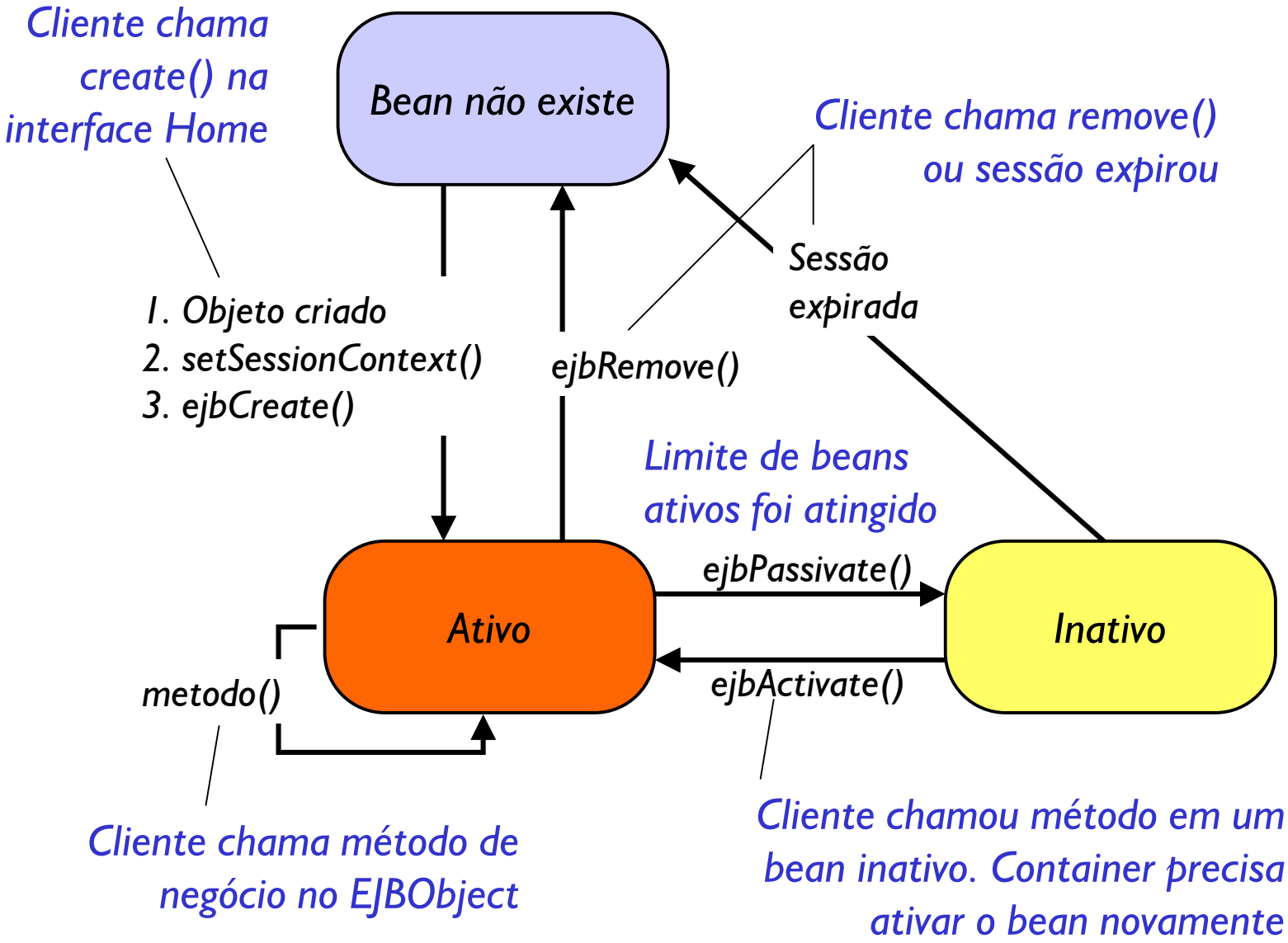
Métodos de um Stateful Session Bean

- *Stateful Session Beans precisam cumprir o mesmo contrato que Stateless beans, mas existem mais operações que podem ser implementadas*
- **`void setSessionContext(SessionContext ctx)`**
  - *Mesma função que stateless beans*
- **`void ejbCreate([Tipo argumentos])`**
  - *Deve haver pelo menos um. Argumentos são opcionais. Pode haver vários ejbCreate(), diferenciados pelo nome e tipo de argumentos*
- **`void ejbRemove()`**
  - *Chamado antes de liberar recursos e remover o bean da memória*
- **`void ejbPassivate()`**
  - *Chamado antes de fazer o swap do bean para o disco*
- **`void ejbActivate()`**
  - *chamado depois que estado do bean é recuperado do disc*

Ativação e Passivação

- Se há mais clientes realizando operações contínuas que beans no pool, container pode criar mais instâncias
- Se há mais clientes que beans no pool, mas uma boa parte permanece inativa por certos períodos (cenário realista), container gerencia recursos usando ativação e passivação
  - Administrador do sistema pode configurar o servidor para obter o melhor desempenho.
- Exemplo: Pool com 5 beans ativos (que participam de sessão) é chamado por sexto cliente.
  - 1. Estado do bean que foi usado há mais tempo é gravado em meio persistente. Antes, seu método ejbPassivate() é chamado
  - 2. Se cliente está continuando diálogo iniciado anteriormente, o estado anterior do bean é recuperado do meio persistente e usado para preencher o bean. Depois, ejbActivate() é chamado.

Ciclo de vida: Stateful Session Bean



Fonte: [1]

Exemplo de Stateful Session Bean

- Veja exemplo no subdiretório cap06
  - mejb2: CountBean (comentado)
  - sun: CheckerBean e CartBean
- Para executar use o Ant (configure build.properties)
  - `ant jboss.deploy` (cria o JAR e copia para o deploy no JBoss)
  - `ant run.jboss.client` (executa um cliente de aplicação)
- Execução
  - Para demonstrar o efeito ejbActivate() e ejbPassivate(), o bean pool foi reduzido artificialmente (usando jboss.xml) para uma capacidade máxima de 2 beans
  - Como o cliente cria mais de dois beans, o container terá que passivar e depois ativar os beans (veja na saída da execução do JBoss quando cada método é chamado)
  - Observe a ordem utilizada pelo container para ativar e passivar (veja que ele passiva o bean usado menos recentemente)

Interfaces Remote e Home

```
package examples;

public interface Count extends javax.ejb.EJBObject {

  public int count() throws java.rmi.RemoteException;

}
```

```
package examples;

public interface CountHome extends javax.ejb.EJBHome {

  Count create(int val)
        throws java.rmi.RemoteException, CreateException;

  Count create()
        throws java.rmi.RemoteException, CreateException;

}
```

Enterprise JavaBean

```
package examples;
public class CountBean implements javax.ejb.SessionBean {
    private SessionContext sessionContext;
    public int val;

    public int count() {
        System.out.println("count() chamado");
        return ++val;
    }
    public void ejbCreate(int val) throws CreateException {
        this.val = val;
        System.out.println("ejbCreate(val) chamado");
    }
    public void ejbCreate() throws CreateException {
        this.val = 0;
        System.out.println("ejbCreate() 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;
    }
}
```

ejbCreate() e create()

- Para cada create() em Home deve existir um ejbCreate() no bean.
  - O número e tipo de argumentos deve combinar
  - Cada create() de Home retorna o tipo da interface do componente
  - Cada ejbCreate() do bean retorna void
  - Ambos provocam as mesmas exceções (exceto RemoteException que só é provocada nas interfaces Home remotas)

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>CountEJB</ejb-name>
      <home>examples.CountHome</home>
      <remote>examples.Count</remote>
      <ejb-class>examples.CountBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

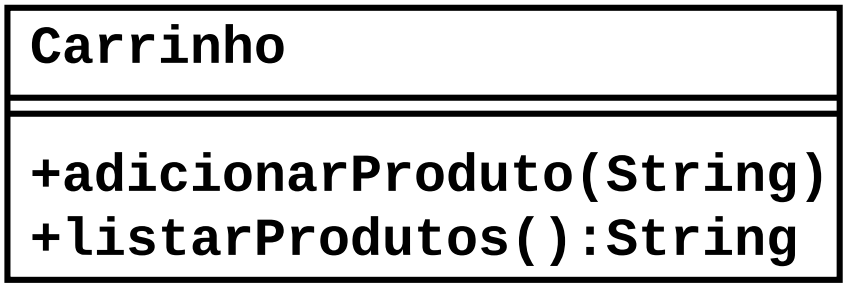
Exercício resolvido em sala (1)

1. Comparação Stateful-Stateless
   - Remova o método create(...) com argumentos do exemplo do CountBean (deixando apenas o sem argumentos).
   - Adapte o cliente para que ele use apenas create() e não create(valor)
   - Execute a aplicação
   - Mude, no deployment descriptor, o bean para Stateless e rode a aplicação novamente. Veja e discuta os resultados.

Exercicio (2)

2. Stateful Session Bean que armazena um String

a) Implemente um session bean com a seguinte interface:



- Bean mantém um string com um produto por linha.
- Listar produtos retorna o string
- Adicionar faz append no string.
- Preencha o ejb-jar.xml e jboss.xml. Faça o deploy.

b) Crie um cliente que

- Crie dois carrinhos
- Preencha ambos com 3 ou 4 produtos diferentes
- Liste o conteudo de ambos.

Exercícios extras (opcionais)

3. Carrinho de compras usando ProdutoVO
   - a) Implemente um Stateful Session Bean que mantenha na memória um HashMap de objetos ProdutoVO.
   - b) O bean deve ter a seguinte interface e utilizar o objeto serializável ProdutoVO

```
public ProdutoVO[] listarConteudoCarrinho();
public void adicionarProduto(ProdutoVO produto);
public void removerProduto(String cod);
public ProdutoVO detalharProduto(String cod);
```

4. Comparação com Stateless
   - Torne seu bean Stateless e veja o que acontece.

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.*

helder@argonavis.com.br

www.argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.5 (junho de 2003)