Java 2 Enterprise Edition

Construção de Aplicações EJB

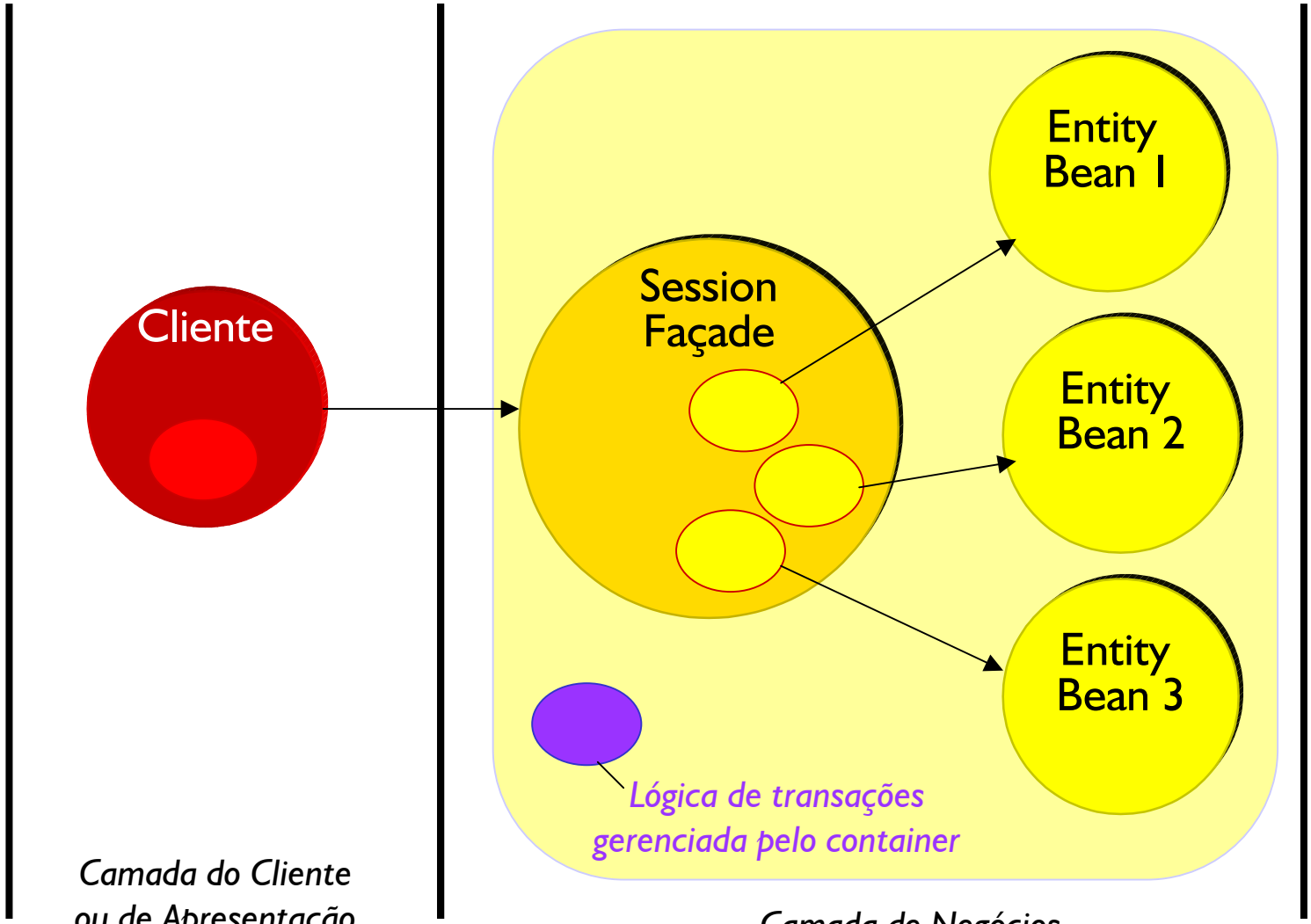
Helder da Rocha
www.argonavis.com.br

Objetivos

- *Este é um módulo prático. O seu objetivo é explorar a criação de uma aplicação EJB*
  - *Comunicação entre beans (relacionamentos)*
  - *Implementação de padrões de projeto como Session Façade, Value Object (Data Transfer Object) e Business Delegate*
- *Os slides apresentados apenas servem como referência. Para o enunciado e exercícios, use os arquivos disponíveis em cap08/exercicios*
- *Exploramos apenas relacionamentos em CMP. Para exemplos com BMP veja projeto do capítulo 15 ou slides do curso J530*

Session Façade

- *Isola beans de entidade dos clientes externos, oferecendo uma fachada com operações suportadas pela aplicação*



Por que implementar?

- Um Session Façade é um padrão bastante comum em aplicações EJB
- Traz diversas vantagens
  - Introduz uma camada controladora entre a camada de negócios e os clientes, centralizando controle de segurança e de transações
  - Expõe uma interface uniforme, mais reduzida porém melhor definida aos clientes, selecionando apenas métodos de interesse
  - Reduz o acoplamento entre cliente e beans
  - Melhora a performance: cliente faz menos chamadas remotas à fachada que chama os beans de entidade via chamadas locais

Como implementar?

- Para implementar um Session Façade é preciso fazer com que o Session Bean possa acessar o Entity Bean como cliente
  - A sintaxe é a mesma utilizada nos clientes comuns
  - A vantagem é que o Session Bean poderá fazer o acesso através das interfaces locais, mais eficientes, e sem a necessidade de usar PortableRemoteObject.narrow()
- Como qualquer cliente, o Session Bean precisará obter uma referência para o bean via JNDI, usar a interface Home para criá-lo e só depois chamar seus métodos
  - Como estão no mesmo container, o session bean pode usar o ENC (java:comp/env) para acessar o bean

Como fazer um bean referenciar outro

- Para que um bean utilize outro bean ele precisa obter uma referência para ele. Isto requer operações na classe do bean e no deployment descriptor
  - No deployment descriptor, defina um EJB Reference e um nome para o bean no domínio java:comp/env
  - No código, localize o bean usando uma chamada JNDI ao domínio java:comp/env/nome_do_bean
- Por exemplo, Session bean que utiliza Produto

```
public class AdminLojaSessionBean implements SessionBean {
  ...
  public String getNomeProduto(String codigo) {
    ProdutoLocalHome produtoHome = (ProdutoLocalHome)
       jndi.lookup("java:comp/env/ejb/loja/produto");
    ProdutoPK pk = new ProdutoPK(codigo);
    ProdutoLocal produto = produtoHome.findByPrimaryKey(pk);
    return produto.getNome();
  }
  ...
```

EJB References

- *Beans referenciam outros beans através de suas interfaces locais ou remotas*
  - *Portanto, há dois tipos de tags para descrever referências EJB de um bean: <ejb-ref> e <ejb-local-ref>*
  - *A referência deve incluir os nomes das classes dos beans e o nome do ENC (java:comp/env) através do qual o bean irá ser referenciado dentro do código*

```
<session>
   <ejb-name>AdminLojaSessionEJB</ejb-name>
   <home>loja.ejb.AdminLojaSessionHome</home>
        ...
   <ejb-local-ref>
      <ejb-ref-name>ejb/loja/produto</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <local-home>loja.ejb.ProdutoLocalHome</local-home>
      <local>loja.ejb.ProdutoLocal</local>
      <ejb-link>ProdutoEJB</ejb-link>
   </ejb-local-ref>
```

Referências de ambiente

- *Assim como referências para returmas (bancos de dados, filas JMS) e outros EJBs, cada bean pode definir um conjunto de variáveis de ambiente, que são acessíveis no seu domínio java:comp/env através do tag <env-entry>*

```
<env-entry>
     <env-entry-name>email/remetente</env-entry-name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>pedidos@loja.com</env-entry-value>
</env-entry>
<env-entry> ... </env-entry>
...
```

- *Environment entries são recuperáveis, dentro do bean, usando JNDI e fazendo o cast para o tipo declarado*

```
public MyBean implements SessionBean {
   String remetente = (String)
         jndi.lookup("java:comp/env/email/remetente");
```

Relacionamentos em EJB

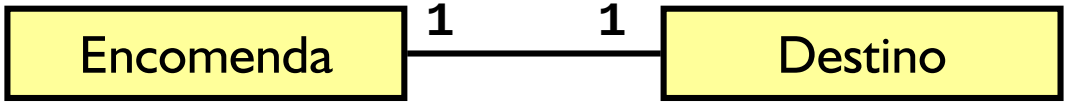
- Como EJB separa a lógica da aplicação dos métodos de controle de persistência e ciclo de vida, implementar relacionamentos fica mais fácil
  - Toda a lógica de relacionamentos pode ser implementada em métodos de relacionamentos que não fazem nenhuma referência ao meio persistente
  - Os métodos de sincronização com o banco (ejbLoad(), ejbStore()) precisam garantir a disponibilidade dos objetos que fazem referência a objetos externos
  - Usando CMP, toda a lógica de relacionamentos é declarativa, o que a torna altamente flexível: pode-se alterar relacionamentos entre objetos sem mexer no código compilado.

Cardinalidade

- Cardinalidade determina quantas instâncias participam de um relacionamento. Há três possibilidades
  - Relacionamentos um-para-um (1:1), por exemplo, uma casa possui um endereço, e um endereço identifica exatamente uma casa
  - Relacionamentos um-para-muitos (1:N), por exemplo, um banco pode ter várias contas, mas cada conta está associada a apenas um banco
  - Relacionamentos de muitos-para-muitos (M:N), por exemplo, uma turma pode ter vários alunos e cada aluno pode participar de várias turmas

Relacionamentos 1:1

- *Representação como objetos*



- *São tipicamente implementados com um relacionamento de chave estrangeira em um banco de dados.*
- *Possível implementação em tabelas*

Encomenda

encomendaPK	descrição	destinoFK
300123	Livros	50000

Destino

destinoPK	cidade	cep
50000	Recife	50183-970

Relacionamento 1:1 em CMP - Bean

- *Em CMP, os métodos de relacionamento são abstratos e os campos de dados não são declarados*

```
public class Encomenda implements EntityBean {
   public abstract Destino getDestino();
   public abstract void setDestino(Destino d);
      ....
   public void ejbLoad() {}
   public void ejbStore() {}
}
```

- *Todos os detalhes do relacionamento precisam ser descritos no deployment descriptor*

```
<ejb-jar>
   <enterprise-beans> ... </enterprise-beans>
   <relationships>
   ...
   </relationships>
   <assembly-descriptor> ... </assembly-descriptor>
</ejb-jar>
```

Relacionamento 1:1 em CMP - DD

```
<relationships>
 <ejb-relation>
  <description>Encomenda-Destino</description>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Enc-Dest</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>EncomendaEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>destino</cmr-field-name>
    </cmr-field>
   </ejb-relationship-role>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Dest-Enc</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <cascade-delete />
    <relationship-role-source>
      <ejb-name>DestinoEJB</ejb-name>
    </relationship-role-source>
   </ejb-relationship-role>

 </ejb-relation>
</relationships>
```

Nome deste relacionamento

Nome da primeira metade do relacionamento

Cardinalidade 1:

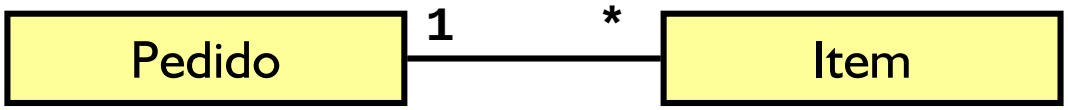
Atributo de Encomenda que permite acesso a destino

Cardinalidade :1

Destino não tem atributo para acessar Encomenda: relacionamento unidirecional!

Relacionamentos 1:N

- *Representação como objetos*



- *Forma mais comum de implementação em tabelas*

Pedido

pedidoPK	status
300123	Confirmado

Item

itemPK	codProduto	qte	pedidoFK
22331	1002	16	300123
22313	9944	1	300166
23112	2943	8	300123

Relacionamentos 1:N em CMP - Bean

- *Mais uma vez, não há nada a fazer em CMP. Basta declar os métodos de relacionamento abstract*

```
public abstract class Pedido implements EntityBean {
   public abstract Collection getItens();
   public abstract void setItens(Collection c);
      ....
   public void ejbLoad() {}

   public void ejbStore() {}
}
```

- *E depois definir os campos de relacionamento dno deployment descriptor*

Relacionamentos 1:N em CMP - DD

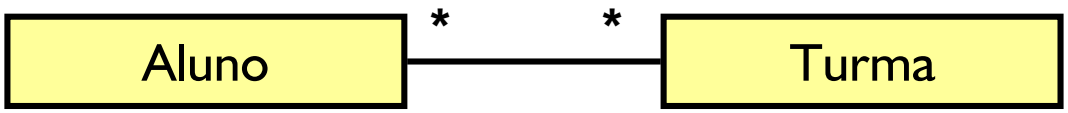
```
<relationships>
 <ejb-relation>
  <description>Pedido-Itens</description>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Ped-Ite</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <relationship-role-source>
      <ejb-name>PedidoEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>itens</cmr-field-name>
      <cmr-field-type>java.util.Collection</cmr-field-type>
    </cmr-field>
   </ejb-relationship-role>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Ite-Ped</ejb-relationship-role-name>
    <multiplicity>Many</multiplicity>
    <cascade-delete />
    <relationship-role-source>
      <ejb-name>ItemEJB</ejb-name>
    </relationship-role-source>
    <cmr-field><cmr-field-name>pedido</cmr-field-name></cmr-field>
   </ejb-relationship-role>
 </ejb-relation>
</relationships>
```

Pode ser Collection ou Set

Um item tem acesso ao seu pedido

Relacionamentos M:N

- *Representação como objetos*



- *São geralmente implementados com dois relacionamentos 1:N. Uma tabela intermediária representa o relacionamento.*
- *Possível implementação*

Aluno

alunoPK	nome
12345	Jasão
99999	Medéia

Turma

turmaPK	curso
40	Java
44	XML

Matricula

matriculaPK	turmaFK	alunoFK
20	40	12345
21	40	99999
22	40	35623
23	41	12345

Estratégias de implementação

- Mapear um bean para cada tabela
  - Fake M:N Relationship - relacionamento M:N falso
  - O resultado desse mapeamento são dois relacionamentos 1:N
  - Se a tabela não tiver informações além das referências o bean só terá campos de relacionamento
- Implementar um mapeamento N:M verdadeiro
  - Útil principalmente se não houver dados para armazenar no objeto intermediário
  - Cada entity beans representa metade do relacionamento e cada um tem uma coleção do outro
- Qual utilizar?
  - Qualquer um, mas o primeiro é mais limpo e fácil de usar

Falso Relacionamento M:N em CMP

```
public class Turma implements EntityBean {
        ....
   public void ejbLoad() {}
   public void ejbStore() {}
}
```

```
public class Aluno implements EntityBean {
        ....
   public void ejbLoad() {}
   public void ejbStore() {}
}
```

```
public class Matricula implements EntityBean {
   public abstract Aluno getAluno();
   public abstract void setAluno(Aluno a);
   public abstract Turma getTurma();
   public abstract void setTurma(Turma c);
        ....
   public void ejbLoad() {}
   public void ejbStore() {} ...
}
```

Falso Relacionamento M:N em CMP - DD (1)

```
<relationships>
 <ejb-relation>
  <description>Matricula-Aluno</description>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Mat-Alu</ejb-relationship-role-name>
    <multiplicity>Many</multiplicity>
    <relationship-role-source>
      <ejb-name>MatriculaEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>aluno</cmr-field-name>
    </cmr-field>
   </ejb-relationship-role>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Alu-Mat</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <cascade-delete />
    <relationship-role-source>
      <ejb-name>AlunoEJB</ejb-name>
    </relationship-role-source>
   </ejb-relationship-role>

 </ejb-relation>
...
```

Falso Relacionamento M:N em CMP - DD (2)

```
...
 <ejb-relation>
  <description>Matricula-Turma</description>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Mat-Tur</ejb-relationship-role-name>
    <multiplicity>Many</multiplicity>
    <relationship-role-source>
      <ejb-name>MatriculaEJB</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>turma</cmr-field-name>
    </cmr-field>
   </ejb-relationship-role>

   <ejb-relationship-role>
    <ejb-relationship-role-name>Tur-Mat</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <cascade-delete />
    <relationship-role-source>
      <ejb-name>TurmaEJB</ejb-name>
    </relationship-role-source>
   </ejb-relationship-role>

 </ejb-relation>
</relationships>
```

Verdadeiro Relacionamento M:N em CMP (1)

```
public abstract class Turma implements EntityBean {
   public abstract Collection getAlunos();
   public abstract void setAlunos(Collection a);
      ....
   public void ejbLoad() {}
   public void ejbStore() {}
}
```

```
public abstract class Aluno implements EntityBean {
   public abstract Collection getTurmas();
   public abstract void setTurmas(Collection c);
      ....
   public void ejbLoad() {}
   public void ejbStore() {}
}
```

Verdadeiro Rel. M:N em CMP - DD

```
<relationships>
 <ejb-relation>
  <description>Turma-Aluno</description>
   <ejb-relationship-role>
    <ejb-relationship-role-name>Alu-Tur</ejb-relationship-role-name>
    <multiplicity>Many</multiplicity>
    <relationship-role-source>
      <ejb-name>Aluno</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>turma</cmr-field-name>
       <cmr-field-type>java.util.Collection</cmr-field-type>
    </cmr-field>
   </ejb-relationship-role>
   <ejb-relationship-role>
    <ejb-relationship-role-name>Tur-Alu</ejb-relationship-role-name>
    <multiplicity>Many</multiplicity>
    <cascade-delete />
    <relationship-role-source>
      <ejb-name>Turma</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name>alunos</cmr-field-name>
       <cmr-field-type>java.util.Collection</cmr-field-type>
    </cmr-field>
   </ejb-relationship-role>
 </ejb-relation>
```

Direção dos relacionamentos

- *Para implementar relacionamentos unidirecionais, exponha os campos de relacionamento em apenas um dos objetos*
  - *Em BMP, apenas um dos objetos deve ter campos e métodos get/set para o relacionamento*
  - *Em CMP apenas um <relationship-role> declara <cmr-field>*
- *Para implementar relacionamentos bidirecionais, exponha campos de relacionamento nos dois beans*
  - *Métodos get/set e campos de relacionamento nos dois beans BMP*
  - *Elementos <cmr-field> nos dois <relationship-role>*
- *A direção dos relacionamentos entre objetos pode não ser igual à direção explícita do esquema do banco de dados*
  - *Esquemas normalizados geralmente indicam um relacionamento unidirecional, mas os objetos têm como obter informações suficientes para implementar qualquer tipo de relacionamento*

Cascade delete em CMP

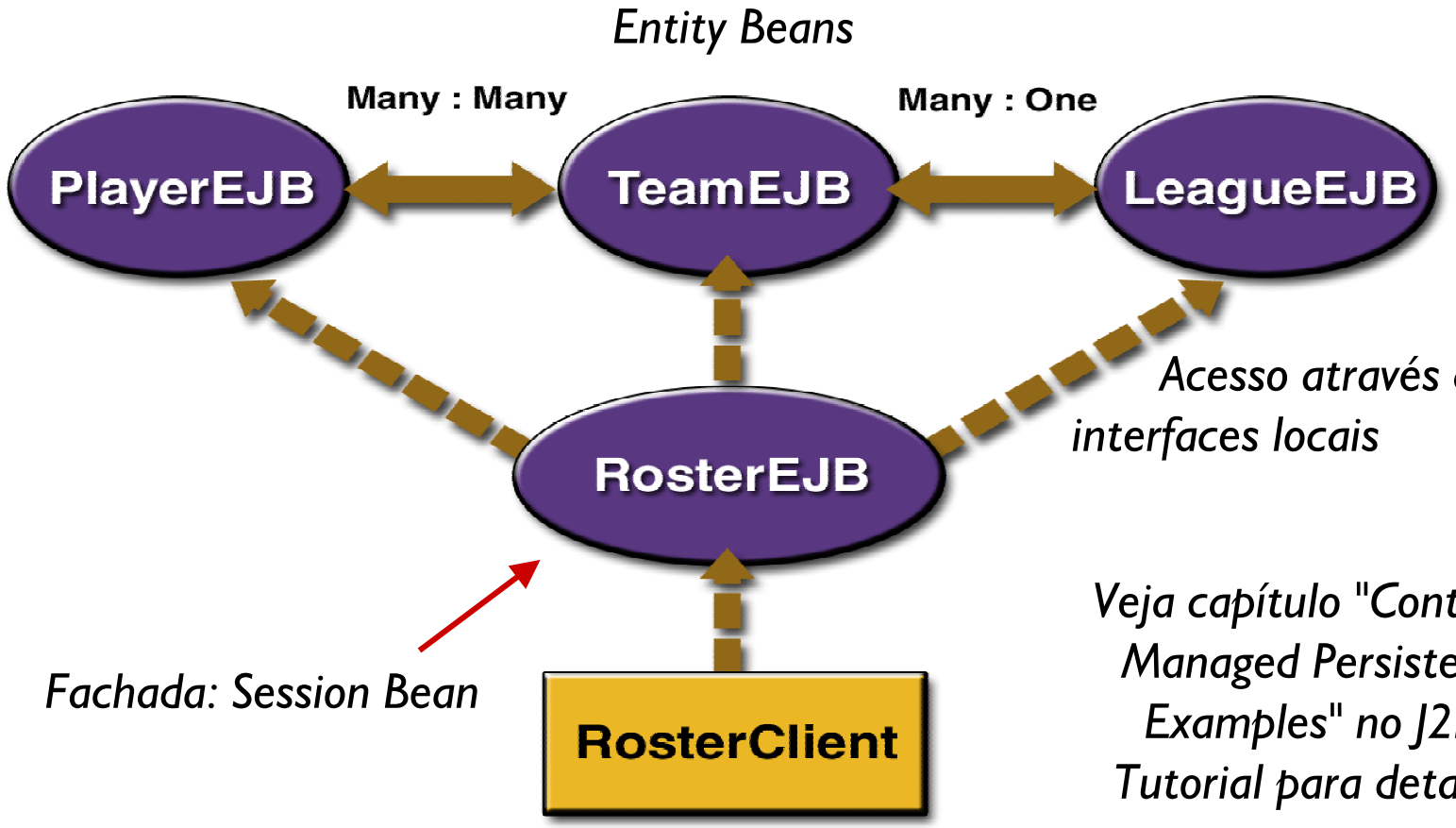
- *Em CMP, o cascade-delete é implementado automaticamente*
  - *Basta incluir o tag <cascade-delete/> no bloco <ejb-relationship-role> de um relacionamento que deve ser removido quando o outro for removido*
- *Exemplo*

```
<ejb-relation>
  <ejb-relationship-role>
    <multiplicity>One</multiplicity>
    <relationship-role-source><ejb-name>Compra</ejb-name></...>
  </ejb-relationship-role>
  <ejb-relationship-role>
    <multiplicity>Many</multiplicity>
    <cascade-delete />
    <relationship-role-source><ejb-name>Item</ejb-name></... >
    <cmr-field><cmr-field-name>pedido</cmr-field-name></...>
  </ejb-relationship-role>
</ejb-relation>
```

Itens serão removidos quando a compra for removida

Exemplo com relacionamentos CMR

- Este exemplo (tutorial da Sun) utiliza relacionamentos gerenciados pelo container (CMR) que requerem
  - Declaração de métodos de acesso ao relacionamento no Bean
  - Especificação dos relacionamentos no deployment descriptor



Declaração de relacionamentos no Bean

- *PlayerBean*

```
public abstract Collection getTeams();          // many teams
public abstract void setTeams(Collection teams);
```

- *LeagueBean*

```
public abstract Collection getTeams();          // many teams
public abstract void setTeams(Collection teams);
```

- *TeamBean*

```
public abstract Collection getPlayers();        // many players
public abstract void setPlayers(Collection players);
public abstract LocalLeague getLeague();        // one league
public abstract void setLeague(LocalLeague league);
```

Veja DD para Team Bean na próxima página

Relacionamentos no DD para TeamBean

```
<relationships>
  <ejb-relation>                    <description>League-Team</description>
    <ejb-relationship-role>
      <ejb-relationship-role-name>LeagueEJB</ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source>
        <ejb-name>LeagueEJB</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>teams</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role>
      <ejb-relationship-role-name>TeamEJB</ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <cascade-delete />
      <relationship-role-source>
        <ejb-name>TeamEJB</ejb-name>
      </relationship-role-source>
      <cmr-field><cmr-field-name>league</cmr-field-name></cmr-field>
     </ejb-relationship-role>
   </ejb-relation>
            ...
</relationships>
```

campo de relacionamento

Remover uma liga causa a remoção de todos os times a ela associados

Como executar

- *Do Sun J2EE Tutorial: RosterApp*
  - *Diretório cap11/exemplos/sun/cmp/*
- *Configuração*
  - *Configure o arquivo build.properties com seu ambiente*
- *Inicialização*

```
> ant drop.table
> ant clean
```

- *Deployment*

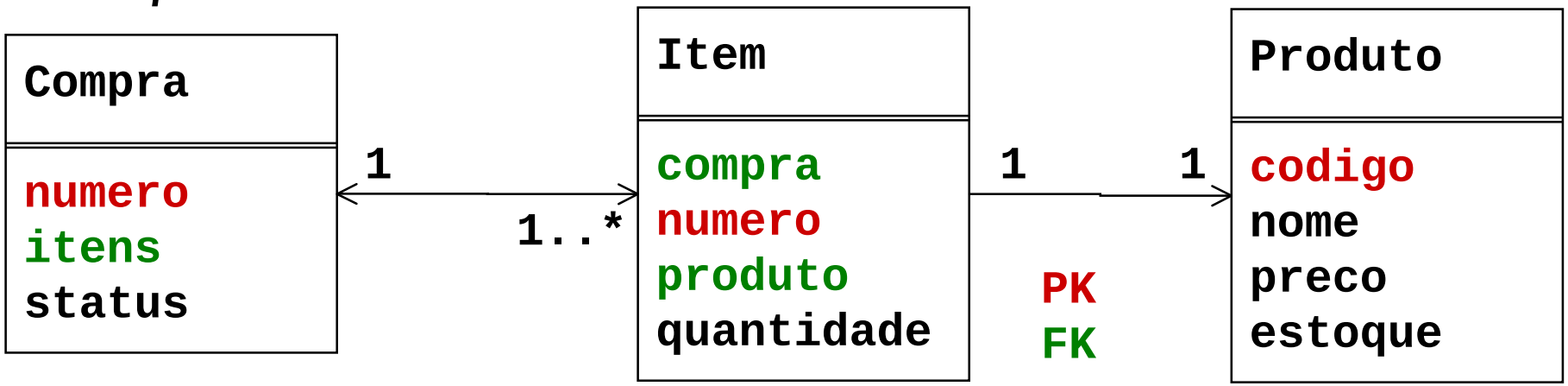
```
> ant jboss.deploy
```
(observe as mensagens de criação das tabelas no JBoss)

- *Execução do cliente*

```
> ant run jboss.client
> ant select.all
```
(faz SELECT na tabela)

Exercícios

- *1. Implemente o seguinte relacionamento entre os objetos Compra, Item e Produto usando CMP*



  - *Uma compra conhece seus itens e cada item conhece sua compra.*
  - *O item conhece seu produto mas o produto ignora a existência de um item*
  - *Garanta que se uma compra for removida, os itens associados a ela também o sejam, mas os produtos não devem ser afetados*
  - *As tabelas estão prontas. Rode ant create-table para criá-las no banco se decidir usar BMP*
  - *Se decidir usar CMP, complete o arquivo jbosscmp-jdbc.xml*

Fontes

[1] *Ed Roman et al. Mastering Enteprise JavaBeans, 2nd. Edition, Chapter 11: CMP & BMP Relationships. John Wiley & Sons, 2002.*

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

[3] *Richard Monson-Haefel. Enterprise JavaBeans, 3rd. Edition. O'Reilly and Associates, 2001*

[4] *J2EE Tutorial. Sun J2EE Tutorial, Sun Microsystems, 2002*

[5] *Emanuel Proulx. EJB Inheritance, Parts 1-4. O'Reilly Network, 2002-2003. www.oreillynet.com/pub/a/onjava/2002/09/04/ejbinherit.html*
(sobre herança em EJB - não abordado nesta versão - veja exemplo do cap. 15 para exemplo de herança entre Entity Beans)

helder@argonavis.com.br

argonavis.com.br

J500 - Aplicações Distribuídas com J2EE e JBoss
Revisão 1.4 (março de 2003)

© 1999-2003, Helder da Rocha