

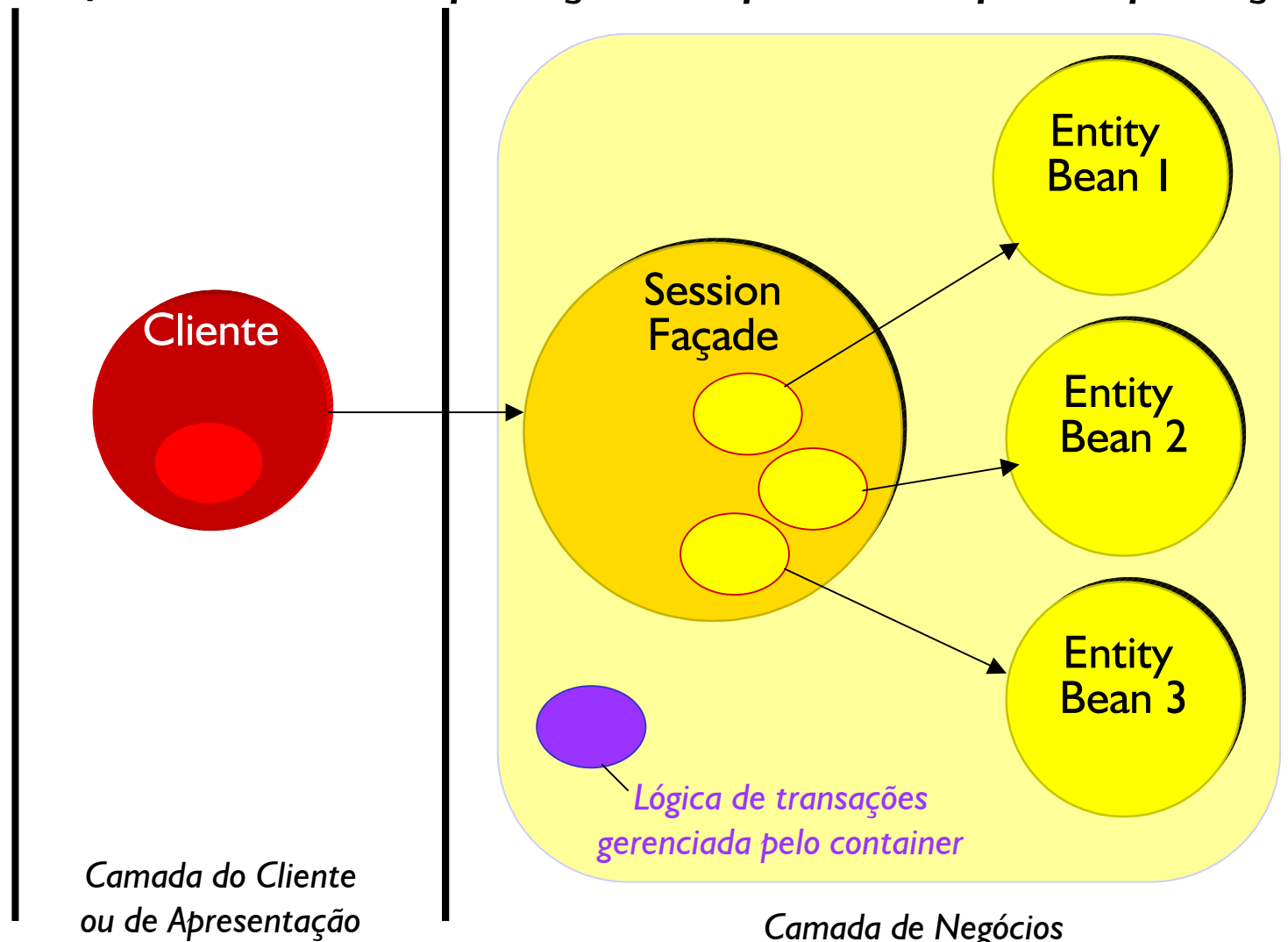
J530 - Enterprise JavaBeans

Relacionamentos em EJB com CMP e BMP

- *Este módulo aborda a modelagem de aplicações orientadas a objetos e sua implementação como Enterprise JavaBeans*
- *Relacionamentos genéricos entre beans*
 - *Session Façade*
 - *Referencias entre beans*
- *Relacionamentos locais entre entity beans*
 - *Relacionamentos 1:1, 1:N e M:N*
 - *Mapeamento de relacionamentos a tabelas*
 - *Direção de relacionamentos*
 - *Associação, composição, integridade referencial e cascading deletes*
 - *Estratégias de implementação*

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 (DD)
 - **No DD**, defina um `<ejb-ref>` (referência JNDI no ENC) para o bean no domínio `java:comp/env`
 - **No código**, localize o bean usando uma chamada JNDI ao domínio `java:comp/env/nome_da_referência`
- 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 interfaces 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 recursos (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 entre entidades

- Quando falamos de relacionamentos geralmente nos referimos a relacionamentos entre entidades
- Em EJB, relações desse tipo ocorrem entre entity beans
 - Os mesmos tipos de relacionamentos existentes entre objetos locais Java comuns (representados, por exemplo, em diagramas UML) podem ser implementados com Entity Beans
- Relacionamentos são implementados em Java através de relações de composição, agregação e associação, uni- ou bi-direcionais, com ou sem regras de integridade referencial
 - Um objeto pode ter uma referência fraca para outro objeto (*associação*: cliente **usa** o outro objeto)
 - Um objeto pode ter uma referência fraca que agrega algo a si (*agregação*: cliente **tem** outro objeto)
 - Um objeto pode ter uma referência sem a qual não poderia existir (*composição*: cliente **tem** um objeto e ele é **parte essencial** sua)

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.

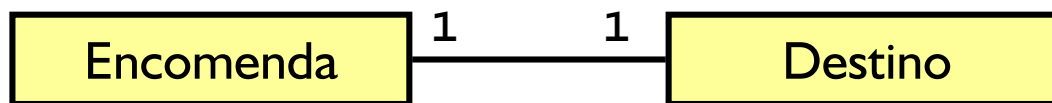
Relacionamentos entre objetos

- Neste módulo, discutiremos relacionamentos entre **objetos**
 - Não tratamos de relacionamentos entre **classes**, como herança
 - Para exemplos de herança entre entity beans, consulte os exemplos do capítulo 15 (Laboratório EJB e referências no final)
- No que se refere a relacionamentos entre objetos, há vários parâmetros que podem ser usados para classificá-los
 - Cardinalidade
 - Direção e visibilidade
- Para que um bean possa ter um relacionamento com outro, é precisa ter uma referência para ele no deployment descriptor (<ejb-ref> ou <ejb-local-ref>)
 - Se o relacionamento é **unidirecional**, apenas **um dos beans** precisará dessa referência;
 - Se o relacionamento for **bidirecional**, **cada bean** deve ter uma referência ao outro bean.

- *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

Tabelas, integridade, direção

- O relacionamento em si nada diz sobre direção do relacionamento ou integridade referencial, mas a implementação afeta esses parâmetros
 - Por exemplo, nas tabelas mostradas, uma encomenda conhece seu destino, mas o destino não conhece a encomenda
 - Esta informação pode ser incluída na outra tabela para tornar o relacionamento bidirecional, ou apenas nos objetos
- Dependendo de como as tabelas foram criadas, uma poderá não remover registros sem remover da outra

```
create table Encomenda (  
    encomendaPK integer primary key,  
    descricao varchar(64),  
    destinoFK integer,  
    constraint destino foreign key (destinoFK)  
        references Destino(destinoPK)  
);
```

```
create table Destino (  
    destinoPK integer primary key,  
    cidade varchar(64),  
    cep varchar(12),  
);
```

Relacionamento 1:1 em BMP

```
public class Encomenda implements EntityBean {
    private int encomendaPK;
    private String descricao;
    private Destino destino;

    public Destino getDestino()          { return destino; }
    public void setDestino(Destino d) { destino = d; }

    ....
    public void ejbLoad() {
        /* dao.select() recupera encomendaPK, descricao e
           destinoFK do banco */
        // Guarda descricao e encomendaPK nas variaveis
        // Constroi Destino a partir de destinoFK */
        DestinoHome home = (DestinoHome)jndi.lookup("Destino");
        destino = home.findByPrimaryKey(new DestinoPK(destinoFK));
    }

    public void ejbStore() {
        int destinoPK = destino.getDestinoPK();
        dao.update(encomendaPK, descricao, destinoPK);
    }
}
```

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:

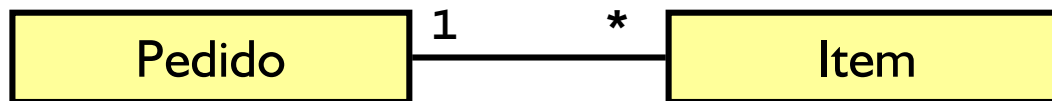
Atributo de Encomenda que permite acesso a destino (get/setDestino())

Cardinalidade :1

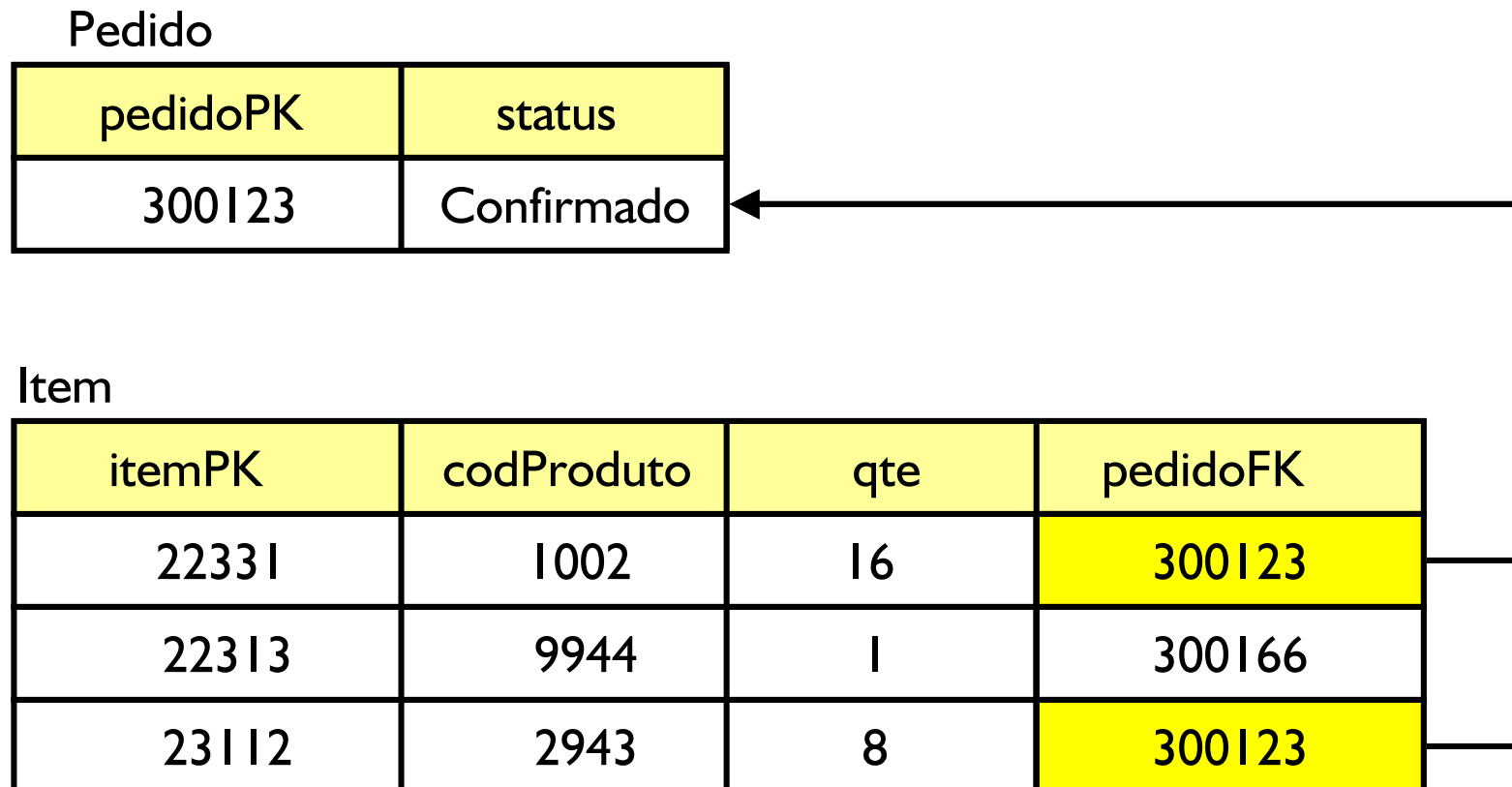
Destino não tem atributo para acessar Encomenda: relacionamento unidirecional!

Relacionamentos 1:N

- *Representação como objetos*



- *Forma mais comum de implementação em tabelas*



Relacionamentos 1:N em BMP

```
public class Pedido implements EntityBean {
    private int pedidoPK;
    private String status;
    private List itens;

    public Collection getItens()        { return itens; }
    public void setItens(Collection c) { itens = c; }

    ....
    public void ejbLoad() {
        // dao.select() recupera pedidoPK e status
        // Guarda status na sua variavel
        // Constroi itens
        ItemHome home = (ItemHome)jndi.lookup("Item");
        itens = home.findByPedido(new PedidoPK(pedidoPK));
    }

    public void ejbStore() {
        dao.update(pedidoPK, status);
    }
}
```

Relacionamentos 1:N em CMP - Bean

- *Mais uma vez, não há nada a fazer em CMP. Basta declarar 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 no 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>

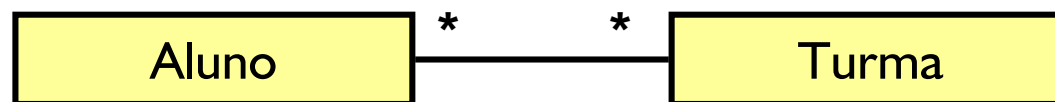
    <ejb-relationship-role>
      <ejb-relationship-role-name>Ite-Ped</ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <cascade-delete />
      <relationship-role-source>
        <ejb-name>ItemEJB</ejb-name>
      </relationship-role-source>
      <cmr-field><cmr-field-name>pedido</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 I: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 BMP (I)

```
public class Turma implements EntityBean {
    private String turmaPK;
    private String curso;

    ....
    public void ejbLoad() {
        // dao.select()
    }
    public void ejbStore() {
        dao.update(turmaPK, curso);
    }
}
```

```
public class Aluno implements EntityBean {
    private String alunoPK;
    private String nome;

    ....
    public void ejbLoad() {
        // dao.select()
    }
    public void ejbStore() {
        dao.update(alunoPK, nome);
    }
}
```

*Classes
inalteradas!*

Falso Relacionamento M:N em BMP (2)

```
public class Matricula implements EntityBean {
    private String matriculaPK;
    private Aluno aluno;
    private Turma turma;

    public Aluno getAluno() { return aluno; }
    public void setAluno(Aluno a) { aluno = a; }
    public Turma getTurma() { return turma; }
    public void setTurma(Turma t) { turma = t; }

    ....
    public void ejbLoad() {
        TurmaHome chome = (TurmaHome)jndi.lookup("Turma");
        AlunoHome ahome = (AlunoHome)jndi.lookup("Aluno");
        // usa dao.select() p/ recuperar chaves estrangeiras
        aluno = ahome.findByPrimaryKey(new AlunoPK(alunoPK));
        turma = chome.findByPrimaryKey(new TurmaPK(turmaPK));
    }

    public void ejbStore() {
        String alunoFK = aluno.getAlunoPK();
        String turmaFK = turma.getTurmaPK();
        dao.update(matriculaPK, alunoFK, turmaFK);
    } ...
}
```

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 (I)

```
<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>
      <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>
    <relationship-role-source>
      <ejb-name>TurmaEJB</ejb-name>
    </relationship-role-source>
  </ejb-relationship-role>

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

Verdadeiro Relacionamento M:N em BMP

```
public class Turma implements EntityBean {
    private String turmaPK;
    private String curso;
    private List alunos;

    public Collection getAlunos() { return alunos; }
    public void setAlunos(Collection a) { alunos = a; }

    ...
    public void ejbLoad() {
        // dao.select()
        alunoHome = (AlunoHome) jndi.lookup("Alunos");
        alunos = alunoHome.findByTurma(new TurmaPK(alunoPK));
    }
    public void ejbStore() {
        dao.update(turmaPK, curso);
    }
}
```

```
public class Aluno implements EntityBean {
    private String alunoPK;
    private String nome;
    private List turmas;

    public Collection getTurmas() { return turmas; }
    public void setTurmas(Collection c) { turmas = c; }

    ...
    public void ejbLoad() {
        // dao.select()
        turmaHome = (TurmaHome) jndi.lookup("Turmas");
        turmas = turmaHome.findByAluno(new AlunoPK(alunoPK));
    }
    public void ejbStore() {
        dao.update(alunoPK, nome);
    }
}
```

Verdadeiro Relacionamento M:N em CMP (I)

```
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

```
<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

- Às vezes não precisamos (ou não devemos) carregar todas as dependências de um bean usando BMP
- Por exemplo
 - Pedido tem um Cliente, que tem um Endereco e uma Collection de Preferencias
 - Listar todos os pedidos vai carregar um monte de informação desnecessária. Isto é "Aggressive Loading"
- Como solução, podemos usar a alternativa de "Lazy Loading" (carga preguiçosa) e carregar apenas a chave primária do cliente
 - Quando um método quiser dados do cliente, pode usar a chave para fazer um `findByPrimaryKey()` e recuperá-lo
- Com CMP, lazy loading acontece automaticamente

Lazy loading: código BMP

```
public class CompraBean implements EntityBean {  
    private String numero;  
    private Status status;  
    private String clienteId;    // bean possui apenas FK - lazy
```

```
    ...
```

```
    public String getClienteId() {  
        return clienteId;  
    }
```

```
    public void setClienteId(String clienteId) {  
        this.clienteId = clienteId;  
    }
```

```
    public Cliente getCliente() {  
        ClienteHome home = (ClienteHome)jndi.lookup("Cliente");  
        return (Cliente)home.findByPrimaryKey(clienteId);  
    }
```

```
    public void ejbLoad() { ... codigo original ... }  
    public void ejbStore() { ... codigo original ... }
```

```
}
```

*Se maior parte dos métodos não
precisa do objeto de relacionamento,
não é preciso recuperá-lo*

*Método que precisar do objeto
completo precisa localizá-lo*



Acoplamento e ordem de remoção

- A remoção de um objeto participante de um relacionamento tem conseqüências que dependem da intensidade do acoplamento do relacionamento
 - Acoplamento forte: **composição** - objeto não pode existir sem o outro (a remoção de um requer a remoção do outro)
 - Acoplamento fraco: **agregação** - objeto independe do outro (a remoção de um não causa a remoção do outro)
- Essas regras podem estar implementadas no esquema do banco de dados, de forma que seria provocada uma exceção caso fossem violadas
 - Para evitar a exceção (ou a violação da integridade dos dados), é preciso implementar o **cascade-delete** (remoção em cascata) para os elementos dependentes, no método `ejbRemove()` do objeto que contém os dependentes

Cascade delete em BMP

- É preciso implementar o **cascade-delete** explicitamente em BMP, e observar a ordem correta de remoção
- Exemplo
 - Uma **Compra** é composta por uma lista de **itens (ItemLocal)**.
 - Removê-la requer a remoção de todos os itens a ela associados

```
public void ejbRemove() {  
    CompraPK pk = (CompraPK)ctx.getPrimaryKey();  
    // 1. Primeiro remove-se os itens de compra  
    try {  
        Iterator it = itens.iterator();  
        while(it.hasNext()) {  
            ItemLocal item = (ItemLocal) it.next();  
            item.remove();  
        }  
    } catch (RemoveException e) { ... }  
    // 2. Depois remove-se o objeto Compra  
    try {  
        compraDAO.remove(pk.numero);  
    } catch (LojaException e) { ... }  
}
```

Remove os beans (registros do banco) que representam itens

Remove o objeto Compra do banco

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

EJB-QL e relacionamentos

- *EJB-QL torna-se bem mais simples que SQL para lidar com relacionamentos*
 - *Não é preciso saber nada sobre colunas ou tabelas*
 - *Pode-se atravessar vários relacionamentos em uma única pesquisa select*
- *Exemplos*
 - `SELECT p.cliente FROM Pedido AS p`
 - `SELECT p.cliente.endereco FROM Pedido p`
- *A pesquisa, porém, é limitada pela navegabilidade entre relacionamentos*
 - *Para ter navegabilidade total, é preciso declarar no deployment descriptor os relacionamentos reversos*

Exemplos com relacionamentos BMP

- Exemplos do tutorial da Sun (adaptados p/ JBoss)
 - Veja em *cap 11/exemplos/sun/bmp*
 - Montar os bancos de dados: **ant create.table**
 - Para fazer o deploy: **ant jboss.deploy**
- Relacionamento BMP 1:1
 - Para rodar: **ant run.one2one.client**



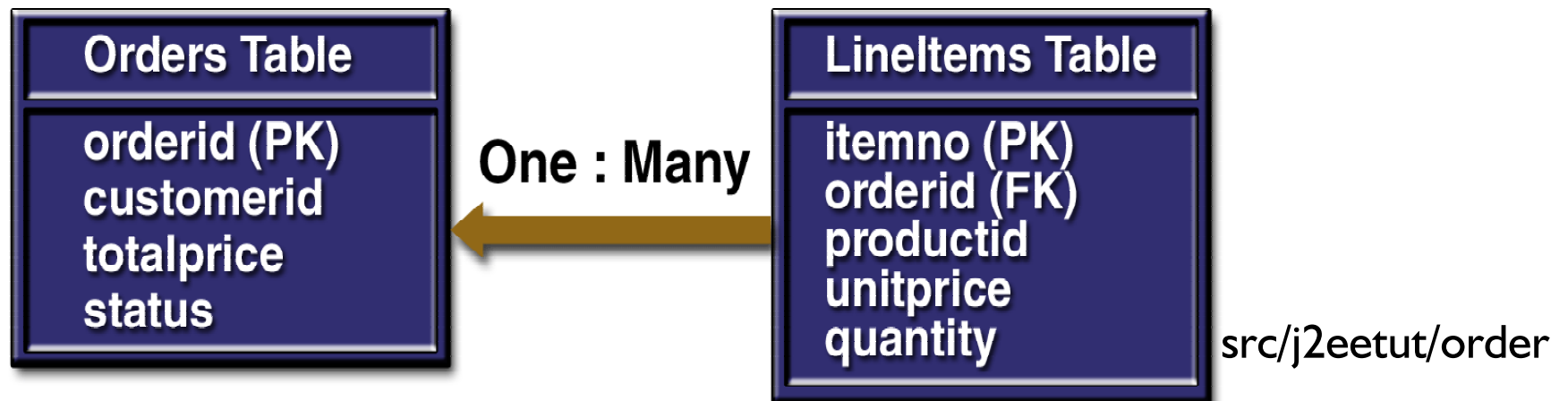
Os relacionamentos são implementados nas tabelas e no código: veja os **scripts SQL**, os **constraints** e também o **código** das interfaces

src/j2eetut/storagebin

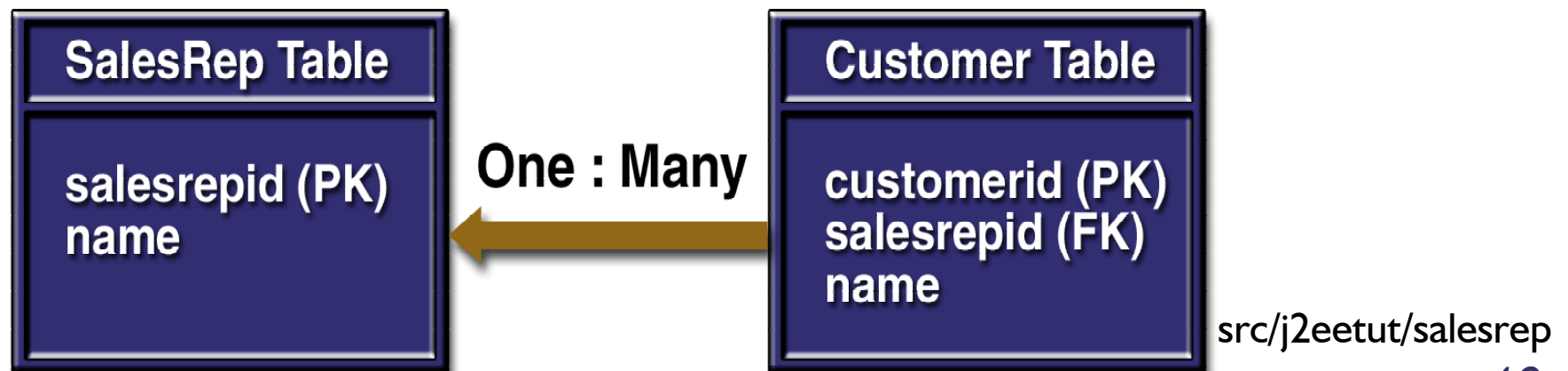
Exemplos com relacionamentos BMP (2)

- Relacionamentos BMP 1:N (dois exemplos)

- Para rodar: `ant run.one2many.helper.client`



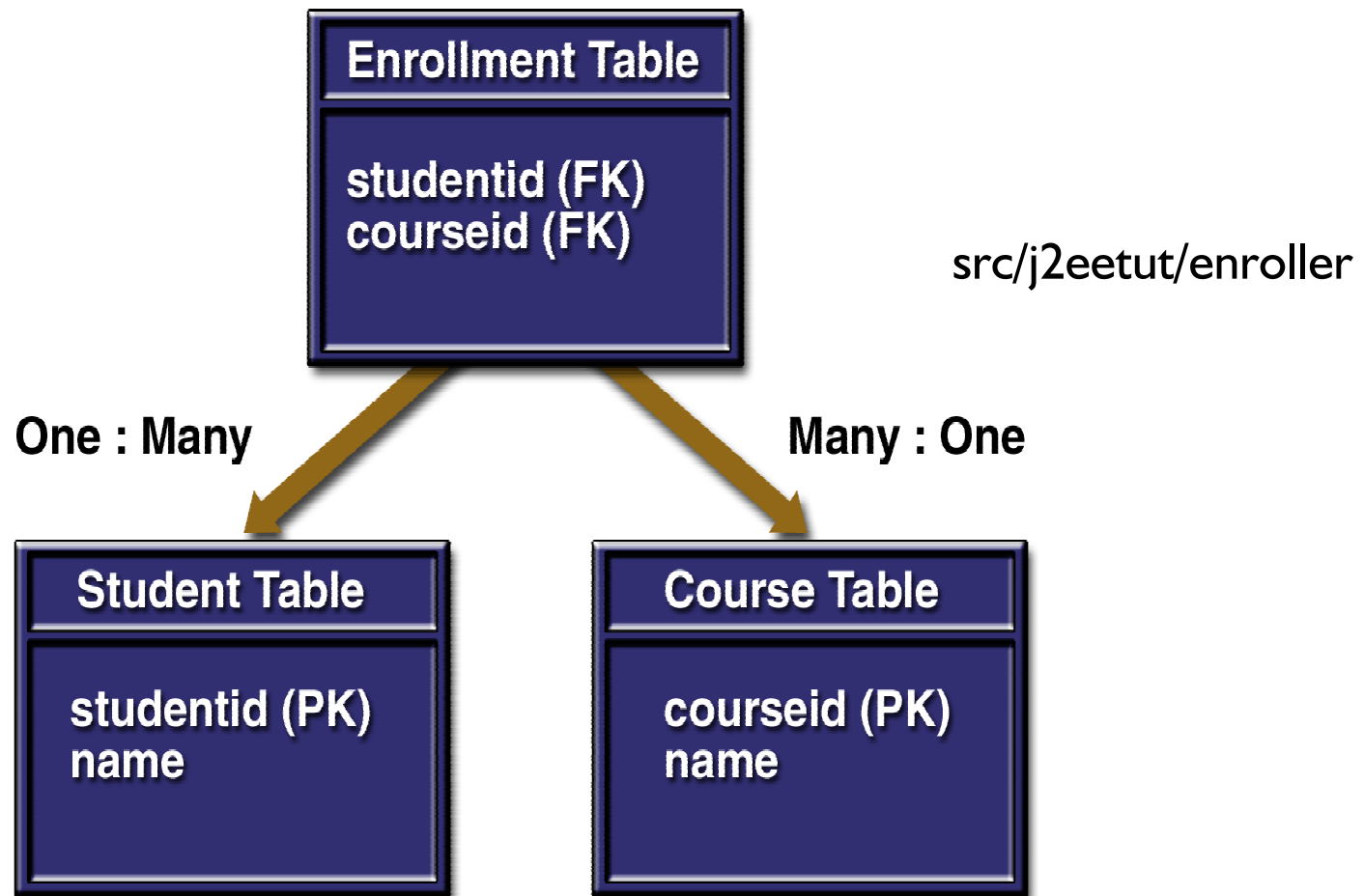
- Para rodar: `ant run.one2many.helper.entity`



Exemplos com relacionamentos BMP (3)

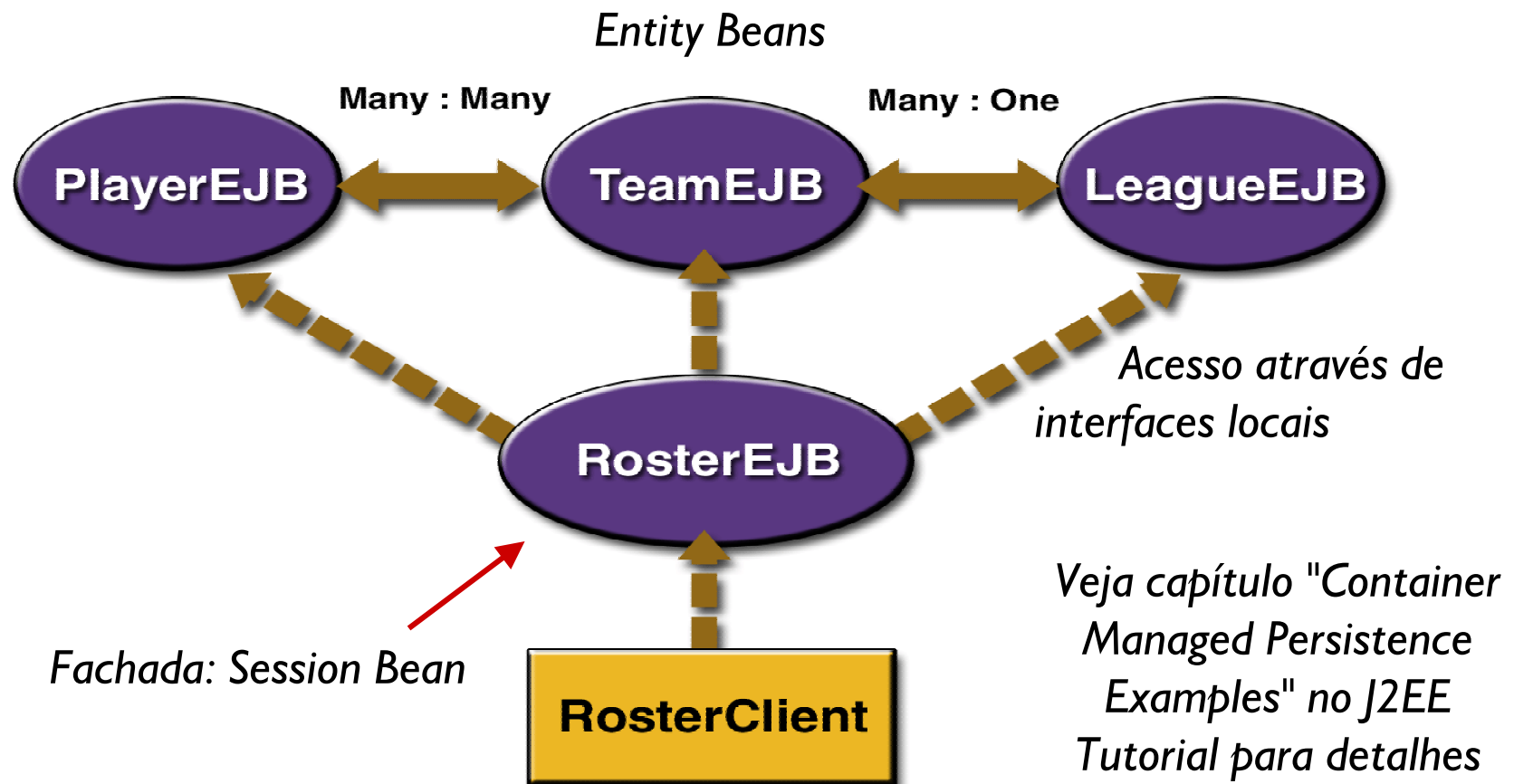
- Relacionamentos M:N

- Para rodar: `ant run.many2many.client`



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);
```

Relacionamentos no DDD 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

- *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*)

1. Transforme o terceiro exercício do capítulo 8 (Entity Beans) para usar Session Façade

- a) Escreva um Session Bean (**ProdutoSession**) para servir de fachada para o **ProdutoBean** com a seguinte interface

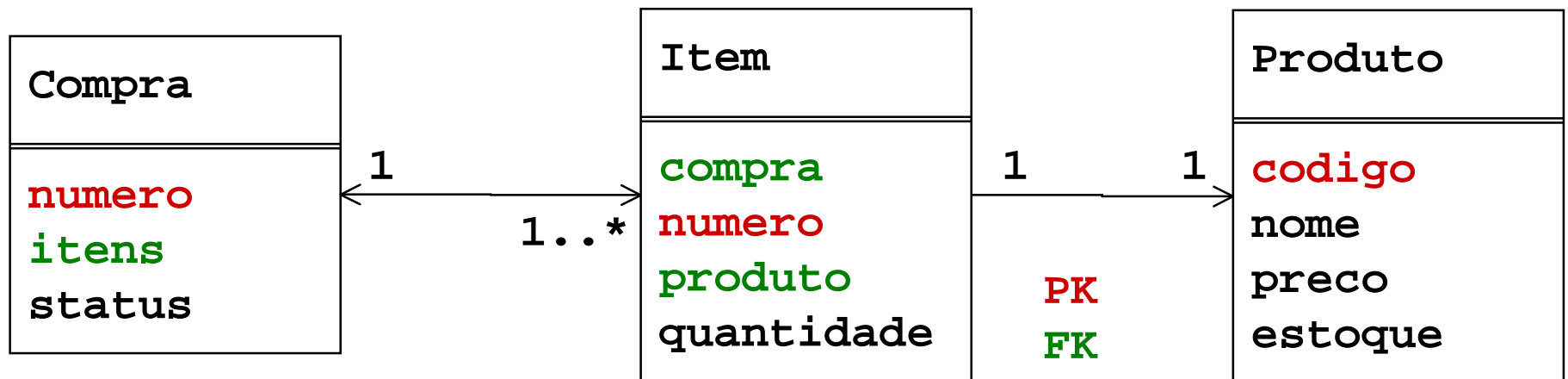
```
void criarProduto(String nome, double preco, int qte)  
String listarProdutos()
```

- b) Escreva um cliente que crie 3 produtos e liste-os através do Session Bean (altere o **build.properties** para que você possa executar o novo cliente através do Ant)
- c) Conecte o **ProdutoSession** ao **ProdutoBean** através de interfaces locais (escreva essas interfaces e defina um **<ejb-local-ref>** no session bean para elas)
- d) Use o mesmo **ejb-jar.xml** para declarar os componentes do entity bean e o session bean.
- e) Configure **jboss.xml** para mapear os nomes JNDI dos dois beans

Exercícios extras (opcionais)

- O código dos objetos dos exercícios abaixo está pronto. Falta implementar os relacionamentos entre as entidades
 - Execute `ant run.client` para testar cada aplicação

2. Implemente o seguinte relacionamento usando CMP



- Uma compra conhece seus itens e cada item conhece sua compra.
- O item conhece seu produto mas o produto ignora o item
- Garanta que se uma compra for removida, os itens associados a ela também o sejam, mas os produtos não devem ser afetados

3. Implemente o relacionamento usando BMP

- As tabelas estão prontas. Rode `ant create-table` para criá-las antes.

- [1] Ed Roman et al. *Mastering Enterprise 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)

Turma J530: Enterprise JavaBeans

Revisão 2.0

© 2001-2003, Helder da Rocha
(helder@acm.org)

 argonavis.com.br