Persistência de Objetos

Objetos Persistentes

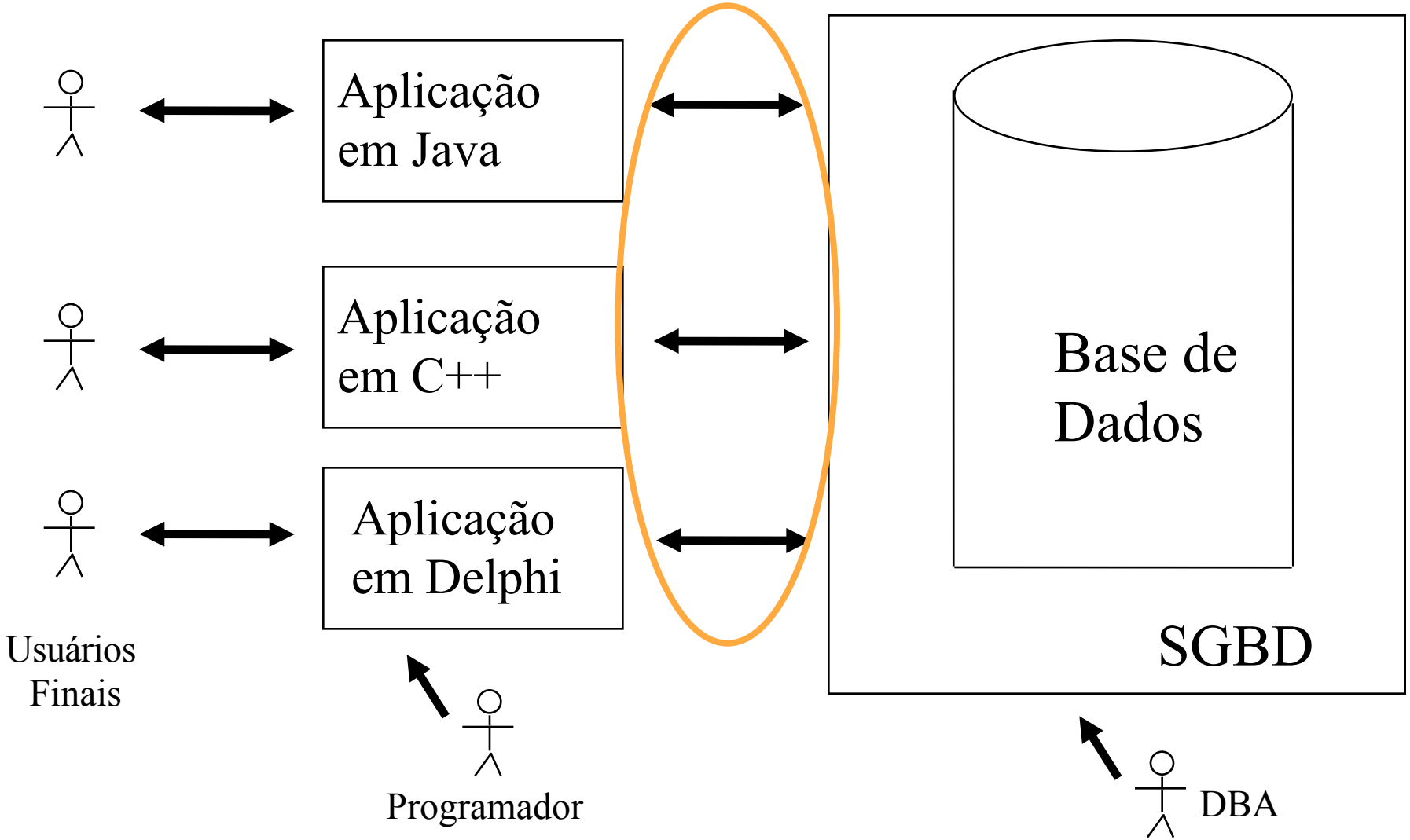
- Objetos Persistentes: são objetos que requerem armazenamento persistente.

Exemplo: Instâncias da classe Descrição Produto devem ser armazenadas em uma base de dados.

Mecanismos de Armazenamento de Objetos

- Banco de Dados Orientado a Objetos
  - ↳ não requer mapeamento para armazenar e recuperar objetos
- Banco de Dados Relacional
- Outros: Banco de Dados Hierárquico, Arquivo, XML, etc.

Armazenamento em um BD Relacional



Persistência de Objetos em SGBD Rel.

↗ Problema da Persistência de objetos em SGBDs relacionais: modelo e linguagem distintos

- Linguagens OO
  - tipos de dados complexos (i.e. classes)
  - hierarquias de herança
  - linguagem imperativa (i.e. métodos)

- SGBDs relacionais
  - tipos de dados simples (domínios atômicos)
  - dados armazenados em tabelas
  - DML declarativa (i.e. SQL)

Persistência de Objetos

- Serviço de Persistência:

Traduz objetos em registros e os salva em uma base de dados, e traduz os registros em objetos quando estes são recuperados de uma base de dados.

Como implementar o serviço de persistência?

1. Através da própria classe de objeto persistente
2. Através de uma camada de persistência

Classe de Objeto Persistente

1. Serviços de Persistência através de uma Classe de Objeto Persistente:

A classe de objeto persistente define o código para salvar e carregar os objetos em uma base de dados.

⇩

Mapeamento Direto

➔ O mapeamento direto precisa ser adicionado e mantido manualmente.

Problemas:

- Alto acoplamento entre a classe e o mecanismo de persistência.
- Serviços técnicos misturados com a lógica da aplicação (baixa coesão).

Camada de Persistência

2. Serviços de Persistência através de uma Camada de Persistência:

Outras classes são responsáveis pelo mapeamento dos objetos persistentes.

Mapeamento Indireto

Para cada classe é definido um mapeador que é responsável por salvar e carregar os objetos da classe na base de dados.

Elementos da Camada de Persistência

1. Fachada

2. Mapeador de Base de Dados

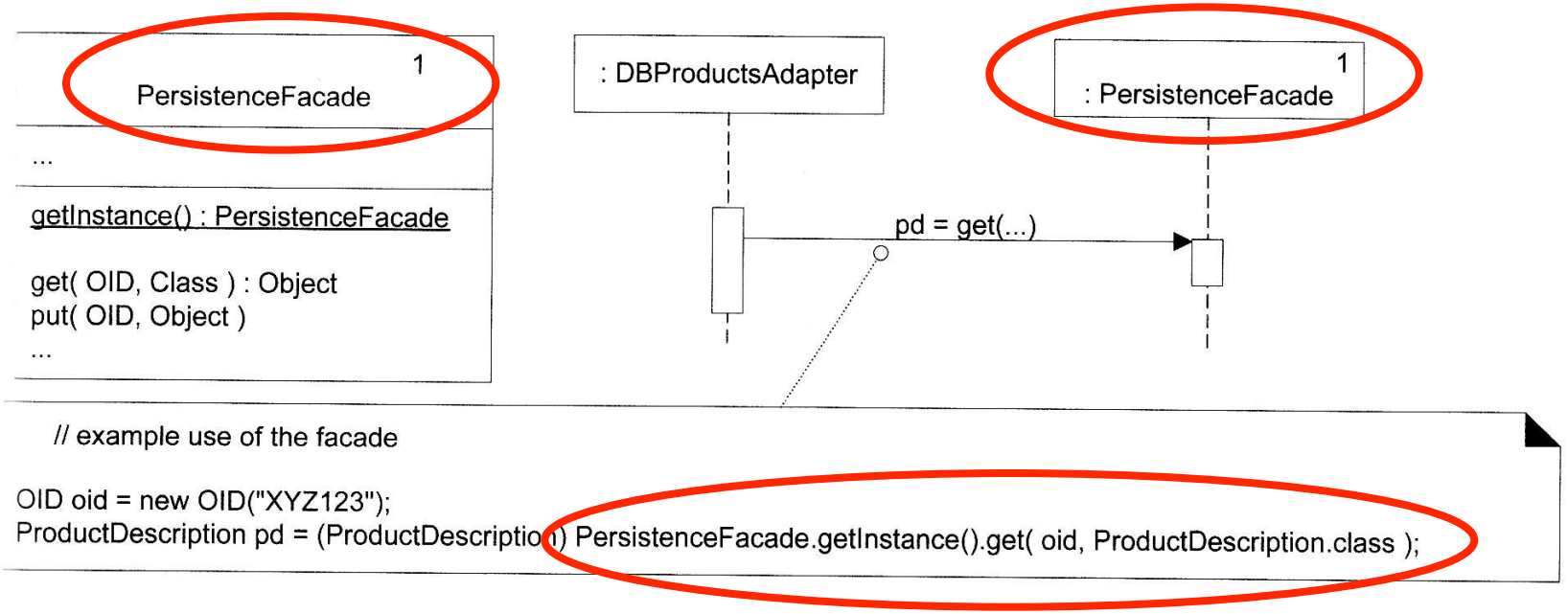
3. Materialização e desmaterialização

4. Caches

Camada de Persistência - Fachada

1. Fachada:

É definida uma fachada para os serviços da camada de persistência.



➡ A operação para recuperar um objeto precisa do OID do objeto e da classe do objeto.

Camada de Persistência - Mapeador

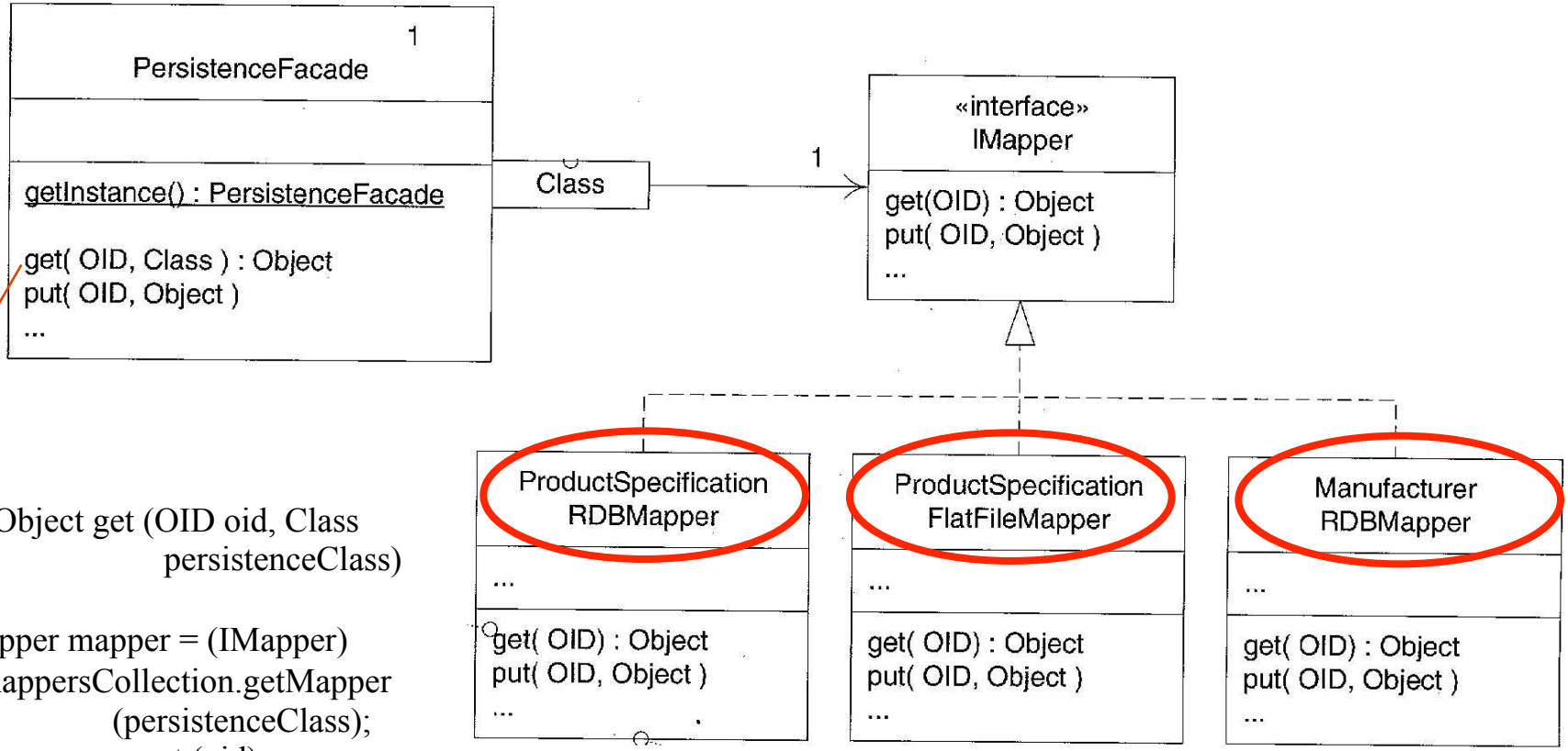
2. Mapeador de Base de Dados:

Como a fachada não faz os serviços de mapeamento, estes serviços são delegados aos mapeadores.

➡ Para cada classe é definido um mapeador que é responsável pela materialização, desmaterialização e caching dos objetos.

Camada de Persistência

- Mapeadores das Classes de Objetos Persistentes




```
public Object get (OID oid, Class
                        persistenceClass)
{
    IMapper mapper = (IMapper)
       mappersCollection.getMapper
                    (persistenceClass);
    return mapper.get (oid);
}
```

Camada de Persistência - Mapeador

Implementação do Mapeadores:

- Mapeadores individuais codificados a mão (código específico)

- 1 Único mapeador baseado em metadados (código genérico)

Gera dinamicamente o mapeamento a partir de um esquema de objeto (metadado que descreve o mapeamento) para a base de dados.

↳ Possível para linguagens com reflexão

Persistência - Materialização

3. Materialização e desmaterialização:

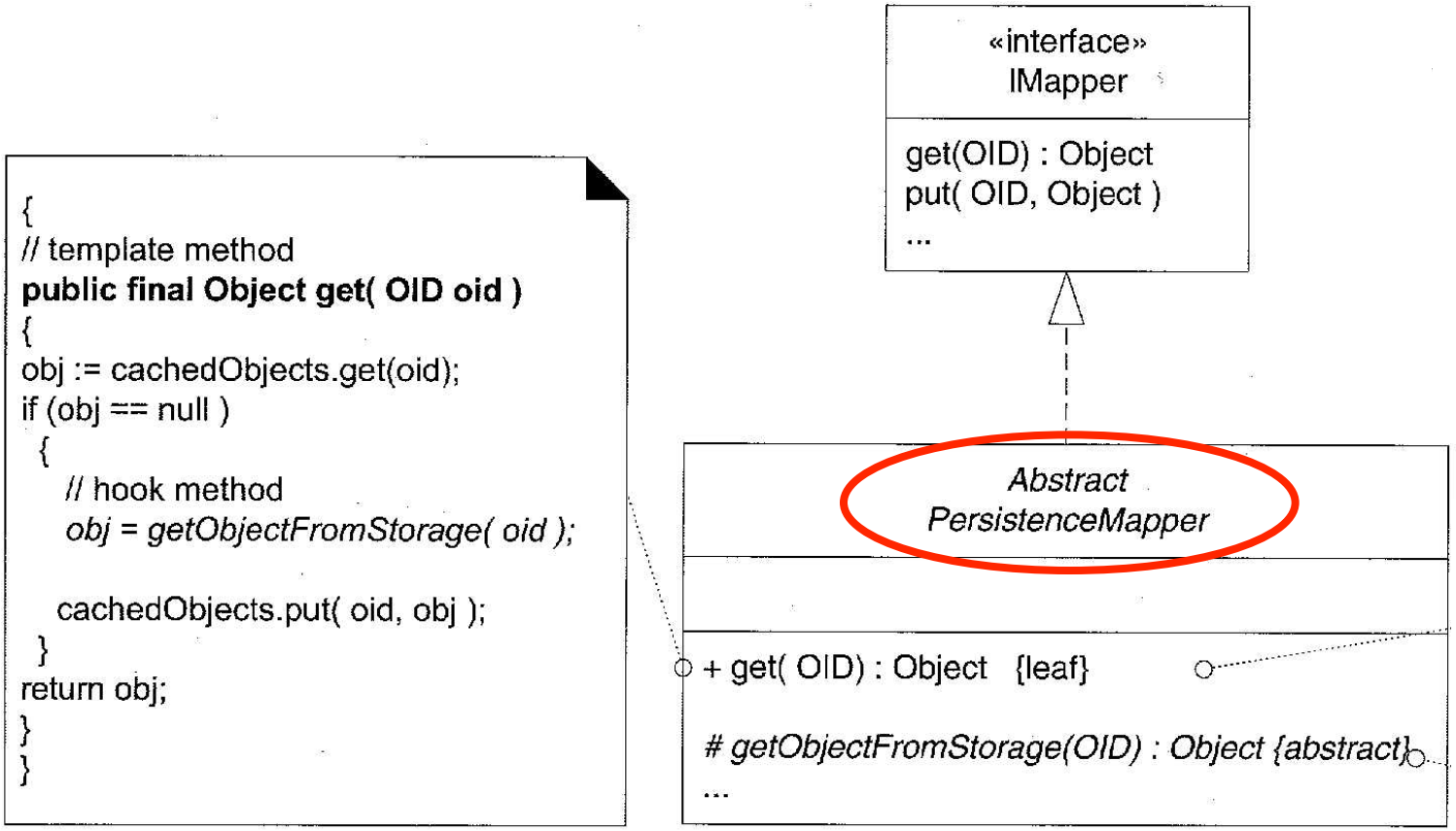
Nos mapeadores são definidos métodos para:

- carregar um objeto da base de dados (método get) - materialização
- salvar um objeto em uma base de dados (método put) - desmaterialização

➡ Como todos os mapeadores apresentam um código comum, este código pode ser definido em uma classe abstrata (método template).

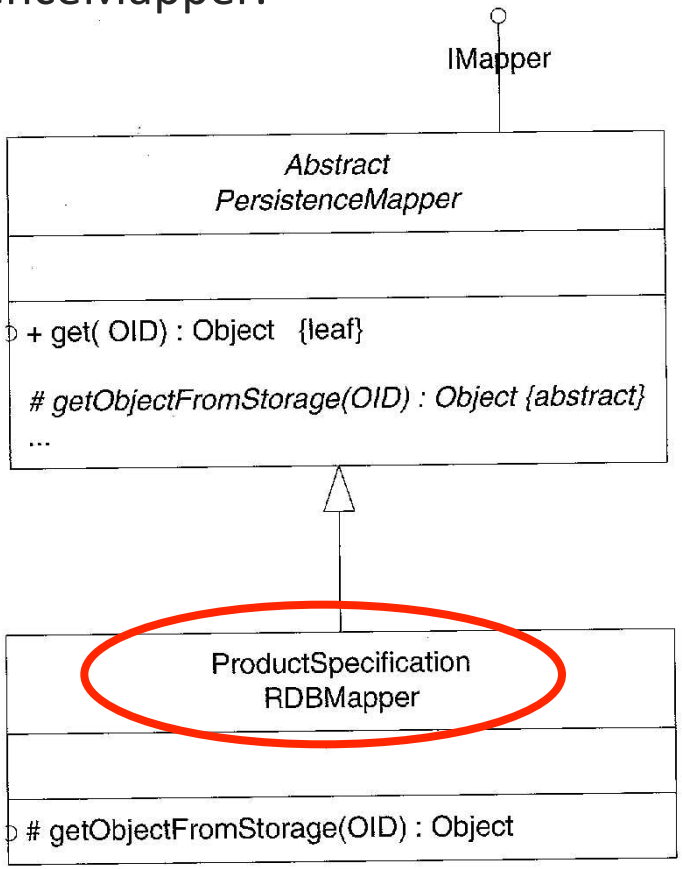
Persistência - Materialização

- Classe Abstrata para o Mapeador



Persistência - Materialização

- Para cada classe de objeto persistente, deve ser definida uma subclasse da classe PersistenceMapper.



Persistência - Materialização

Exemplo do método getObjectFromStorage do mapeador

```
protected Object getObjectFromStorage (OID oid)
{
 String key = oid.toString();
 dbRec = db.executeSql("SELECT PRICE,ITEM_ID,DESC
                        FROM PROD_DESC
                        WHERE OID = " + key);
 ProductDescription pd = null;
 if (dbRec.next()) {
       pd = new ProductSpecification ();
       pd.setOID (oid);
       pd.setPrice (dbRec. getDouble ("PRICE") );
       pd.setItemID (dbRec. getLong ("ITEM_ID") );
       pd.setDescrip (dbRec. getString ("DESC") );
 }
dbRec.close(); // fecha o cursor do result set
 return pd;
}
```

Persistência - Materialização

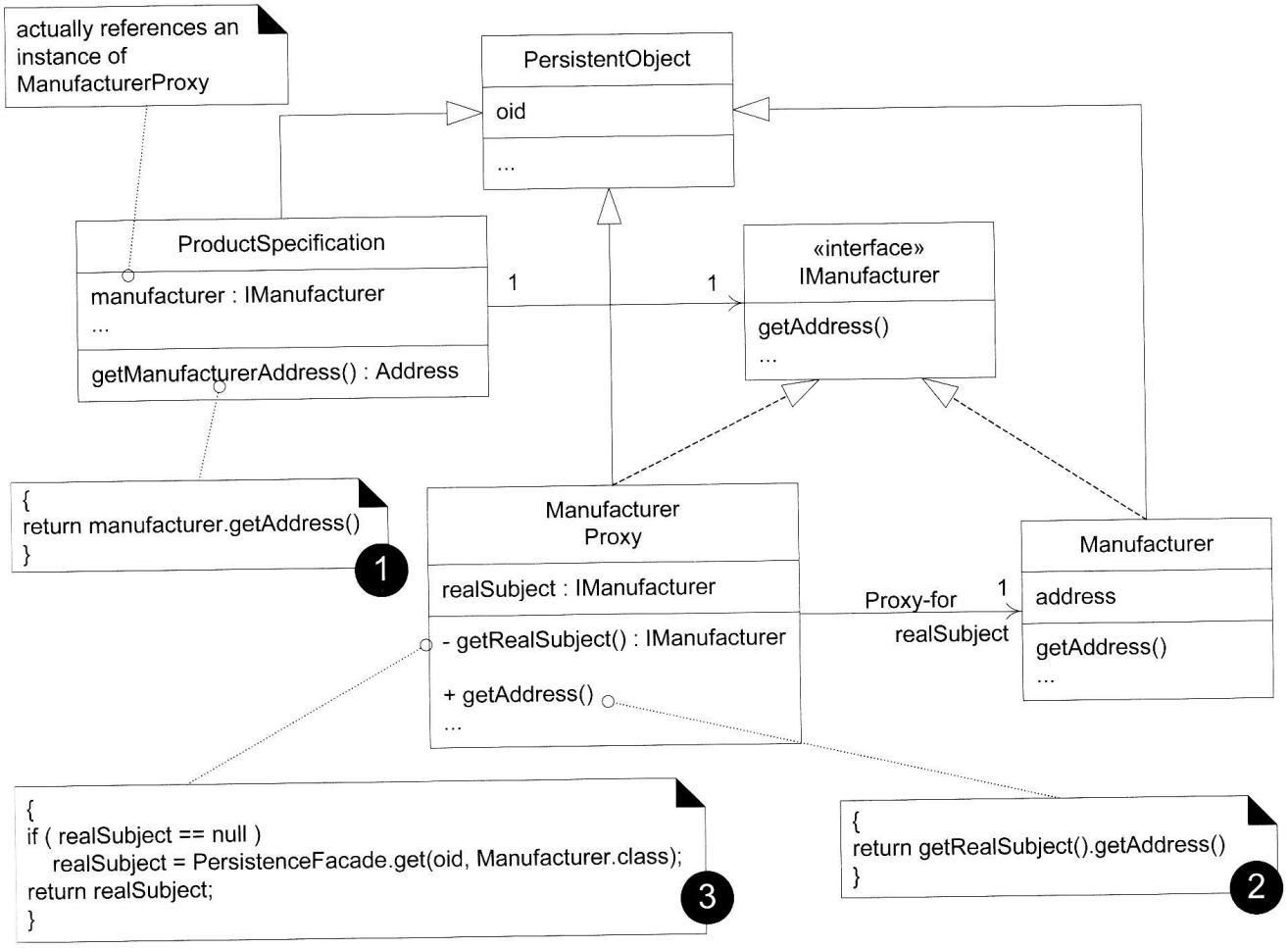
- Lazzy Materialization (Materialização Por Demanda):

Adia a materialização dos objetos referenciados.

Pode ser implementada usando o padrão de projeto Proxy.

Persistência - Materialização

- Lazzy Materialization (Materialização Por Demanda):



Persistência - Cache

- 4. Cache:

Para aumentar a performance, os objetos materializados são mantidos pelos mapeadores das classes em um cache.

↳ quando um objeto é carregado ou inserido na base de dados na primeira vez, ele é incluído no cache.

Mapeamento Estrutural OO - Relacional

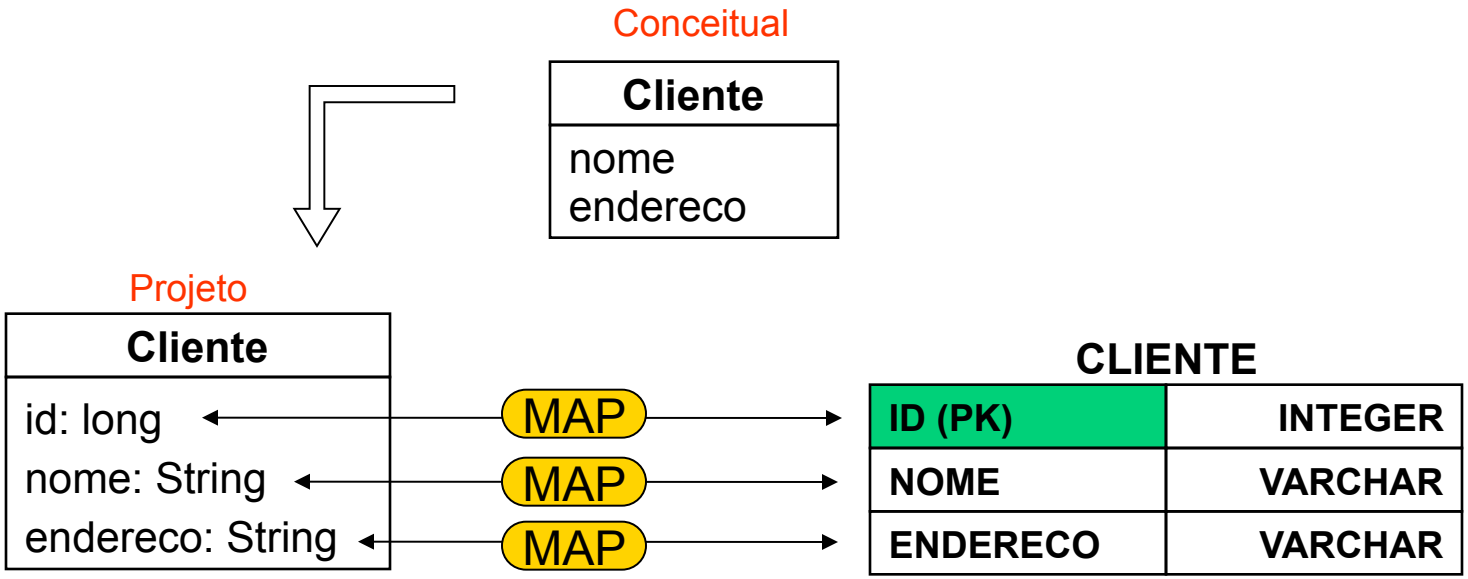
Mapeamento Estrutural

- Classes / Atributos
- Associações
- Herança

Classes - Mapeamento Estrutural

- Para cada atributo mapeado deve existir uma coluna na tabela correspondente cujo domínio é compatível com o tipo do atributo.

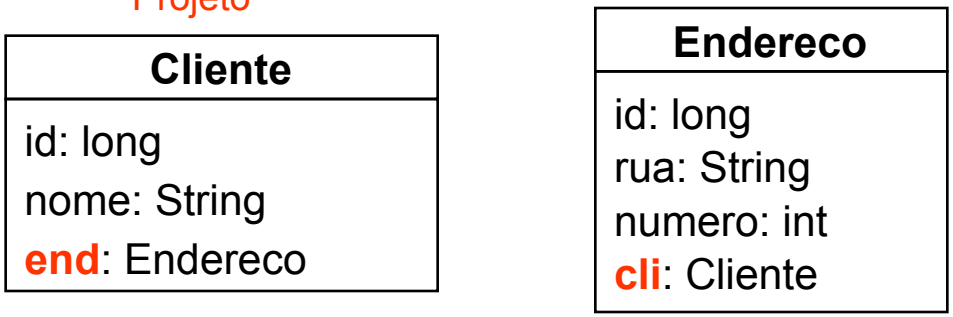
- O atributo identificador da classe deve estar mapeado para a coluna correspondente à chave primária. Caso não exista um atributo identificador na classe original, é necessário criá-lo.



Associações 1:1 - Mapeam. Estrutural

➡ As classes associadas podem ser mapeadas para uma mesma tabela ou tabelas separadas.

Projeto

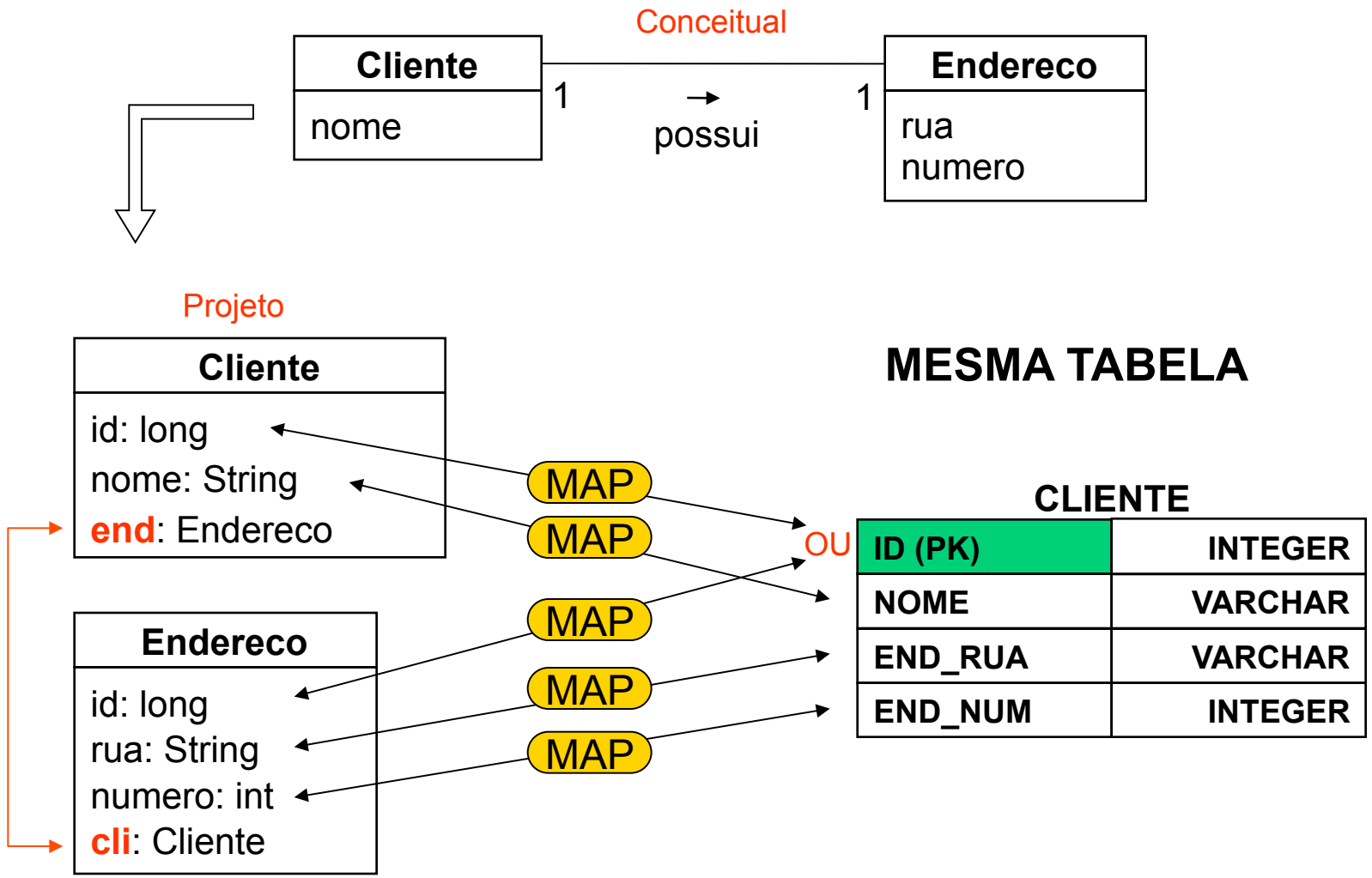


➔ Se as classes são mapeadas para uma mesma tabela, o mapeamento é direto.

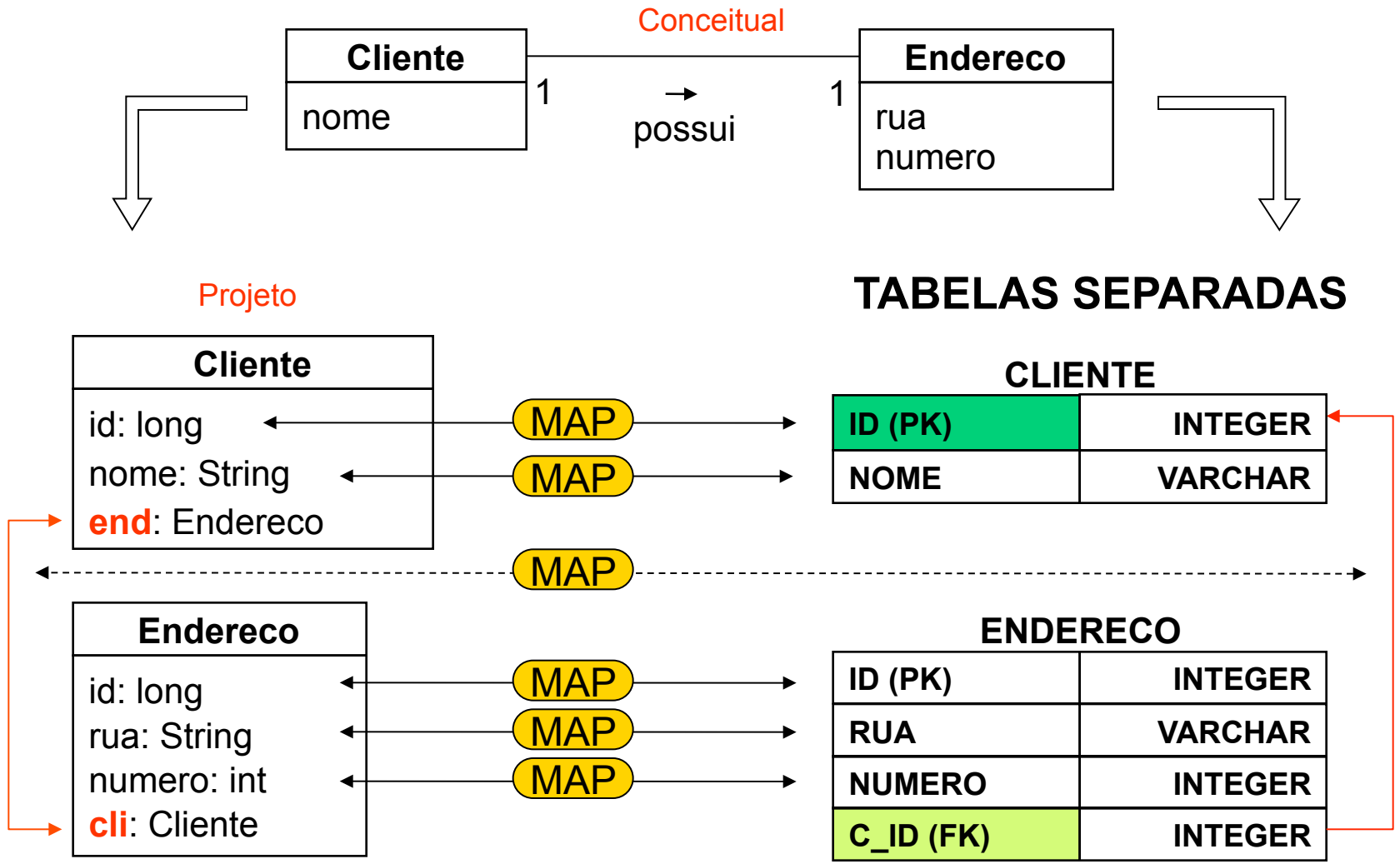
➔ Se as classes mapeiam para tabelas separadas, uma chave estrangeira precisa estar definida em uma das tabelas.

Obs: É conveniente que as classes associadas refiram-se mutuamente através de referências inversas.

Associações 1:1 - Mapeam. Estrutural



Associações 1:1 - Mapeam. Estrutural



Associações 1:N - Mapeam. Estrutural

- As classes associadas devem estar mapeadas em tabelas separadas

- A tabela correspondente à classe do lado N da associação deve conter uma chave estrangeira para a tabela correspondente à classe do lado 1

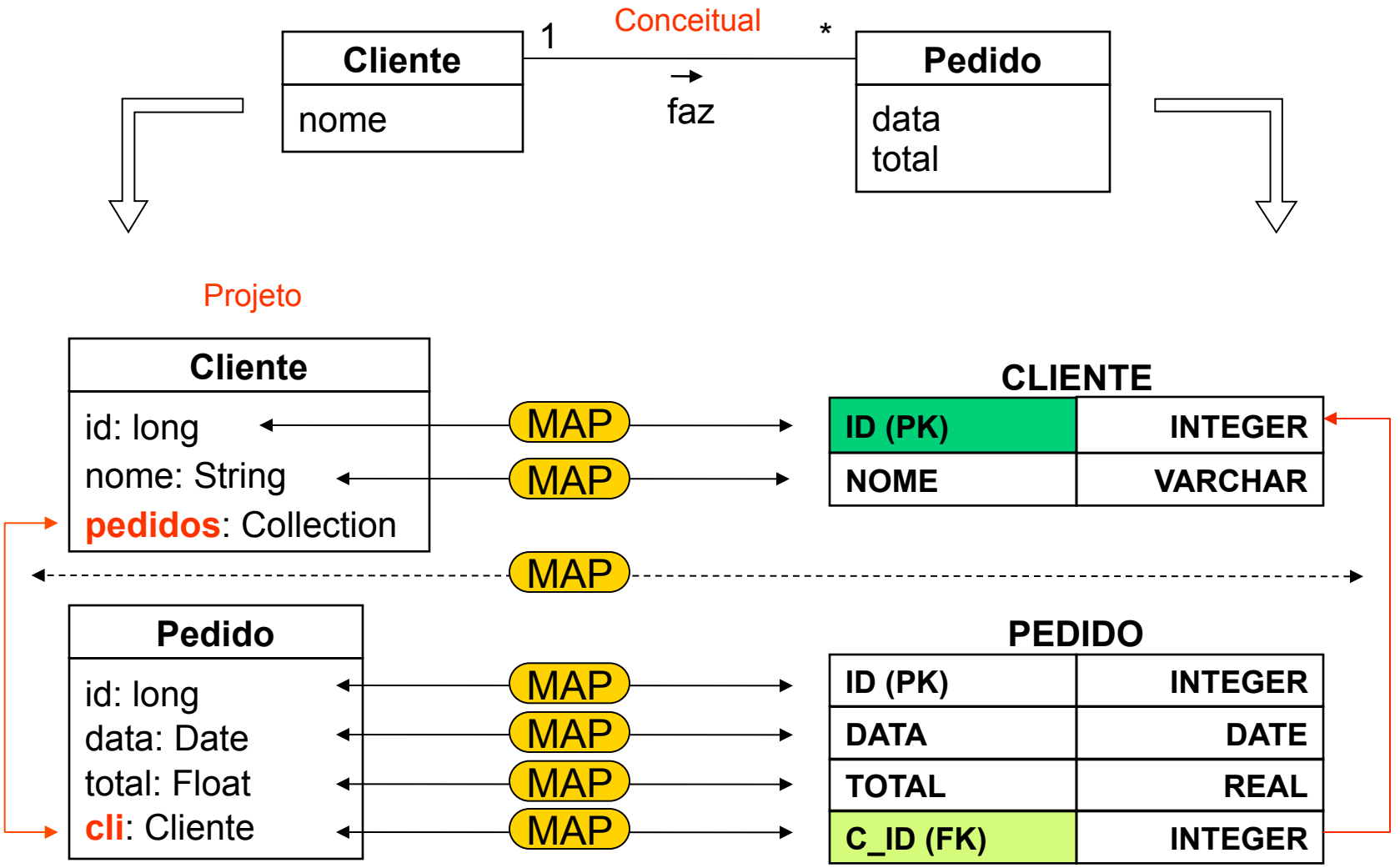
Projeto

Cliente
id: long nome: String **pedidos**: Collection

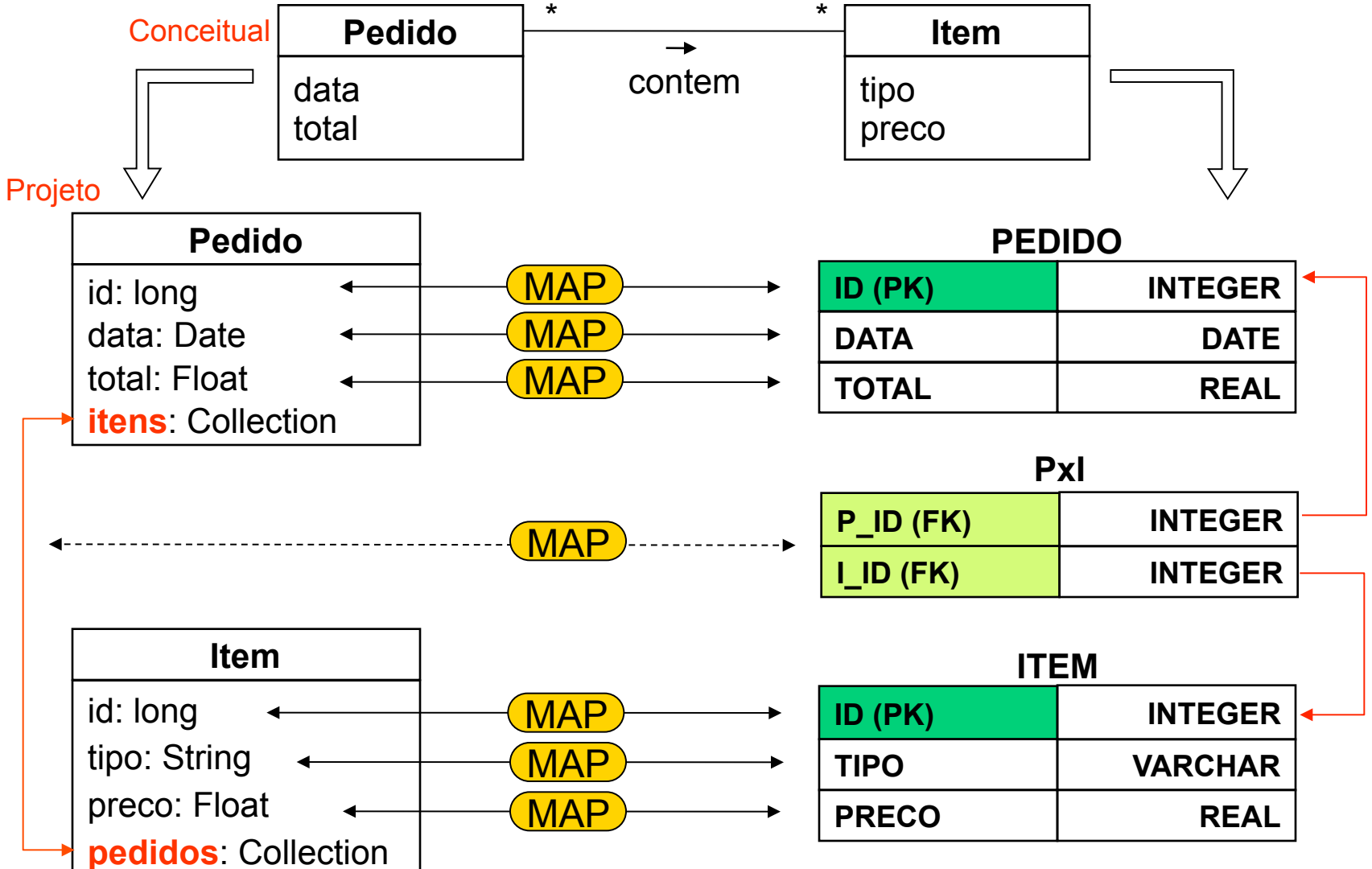
Pedido
id: long data: Date total: Float **cli**: Cliente

Obs: É conveniente que as classes associadas refiram-se mutuamente através de referências inversas

Associações 1:N - Mapeam. Estrutural



Associações N:N - Mapeam. Estrutural



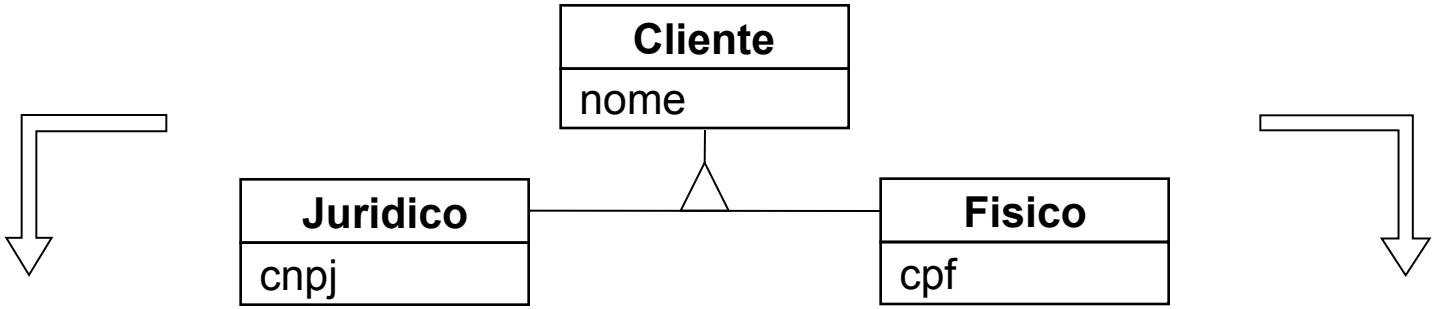
Herança - Mapeamento Estrutural

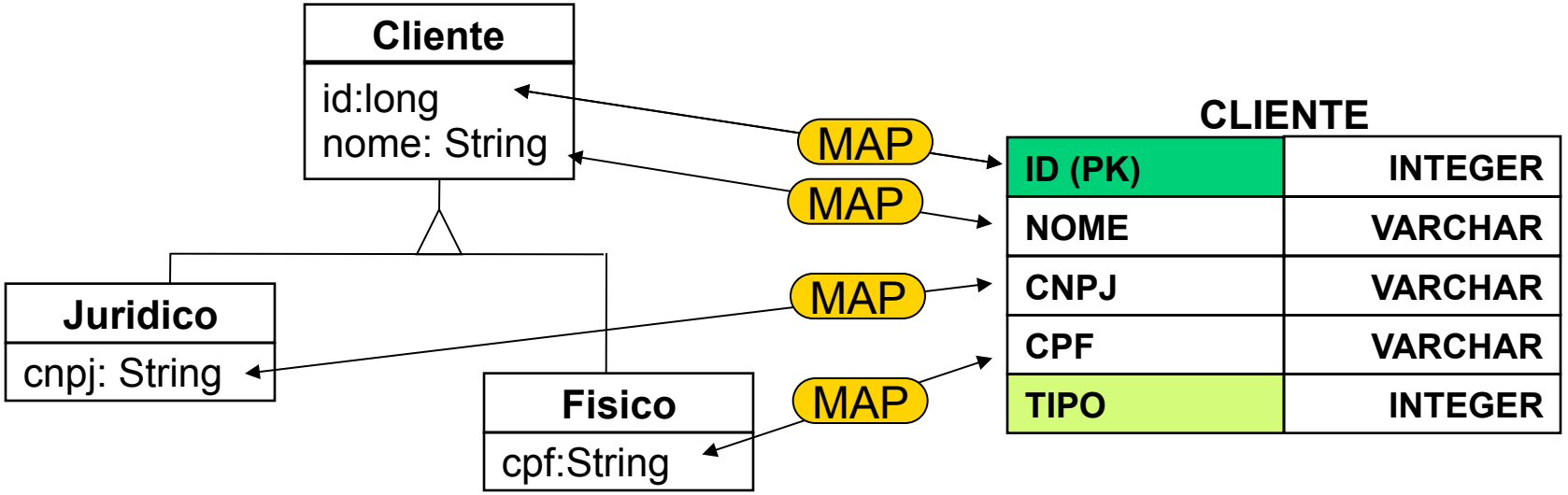
➡ As classes da hierarquia podem estar mapeadas para uma mesma tabela, tabelas separadas, ou cada classe em uma tabela diferente*.

➔ Mesma tabela: se as classes são mapeadas para uma mesma tabela, esta tabela deve conter colunas para todos os atributos da superclasse e respectivas subclasses, além de um atributo discriminador que indica o tipo do objeto.

* Usado em sistemas legados.

Herança - Mapeamento Estrutural





Herança - Mapeamento Estrutural

➡ As classes da hierarquia podem estar mapeadas para uma mesma tabela ou tabelas separadas, ou cada classe em uma tabela diferente*.

➔ Tabelas separadas: se as classes são mapeadas para tabelas separadas, cada tabela deve conter apenas colunas para os atributos da classe correspondente. Além disso, as tabelas das subclasses devem se referir à tabela da superclasse através de uma chave estrangeira. Neste caso, o atributo discriminador fica na tabela da superclasse raiz.

Herança - Mapeamento Estrutural