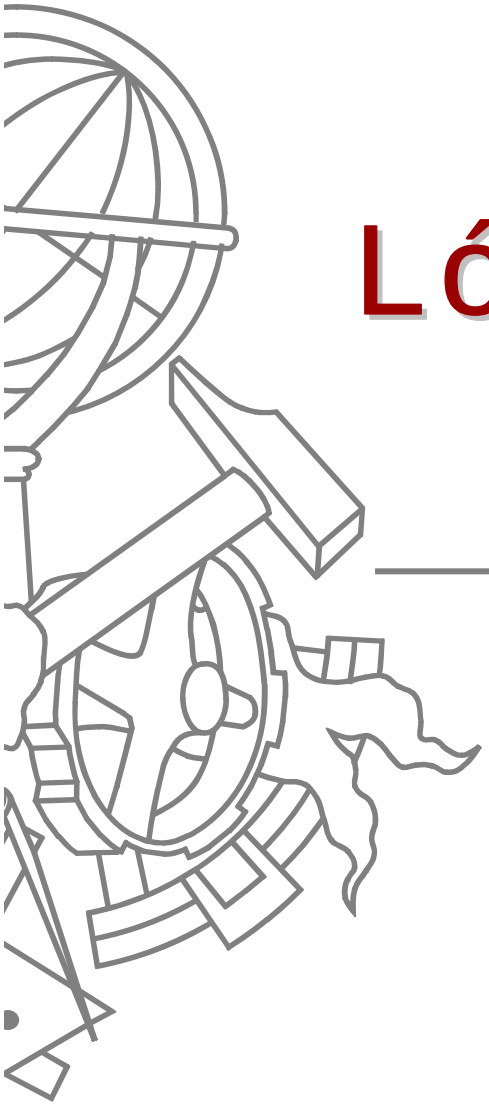
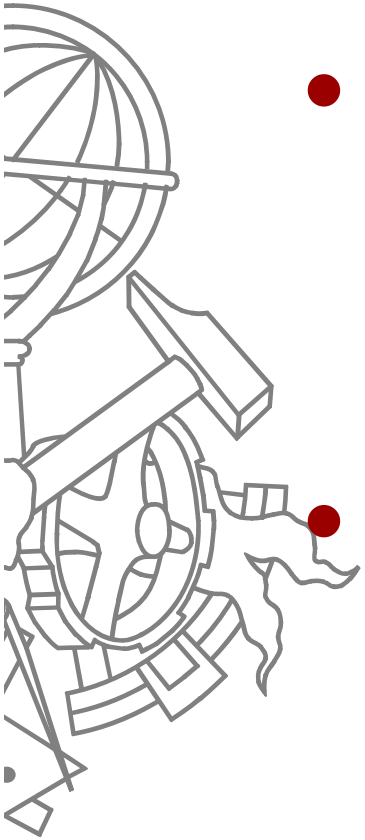


Lógica de Negócio e Acesso a dados

Parte 2.2

Estilos arquitecturais



- Orientado à tabela
  - Table Module
  - Table Data Gateway

BLL

DAL

- Orientado aos objectos
  - Domain Model
  - Active Record
  - Data Mapper

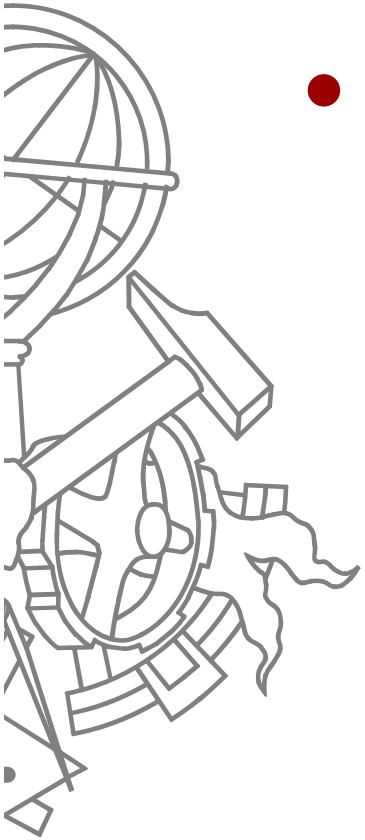
BLL

DAL

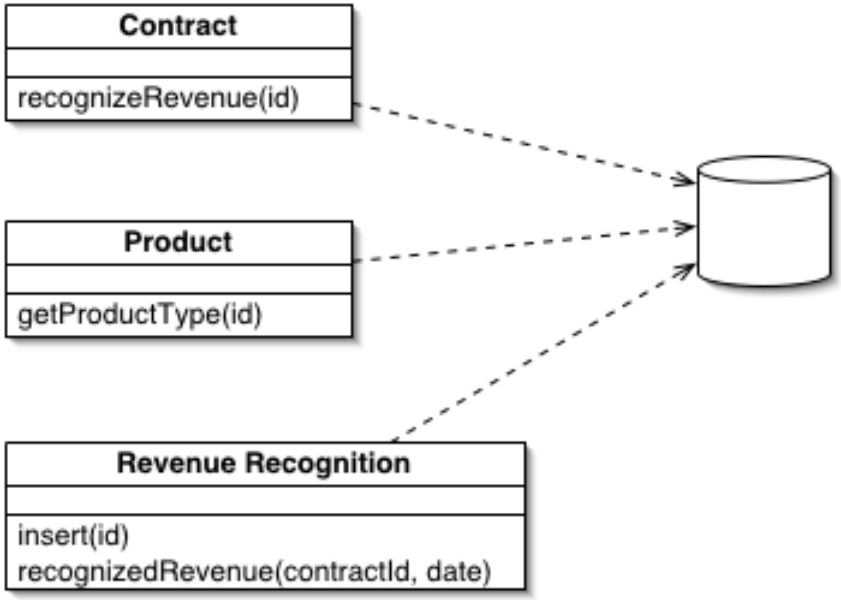
Table Module com Table Data Gateway

Pattern

Table Module



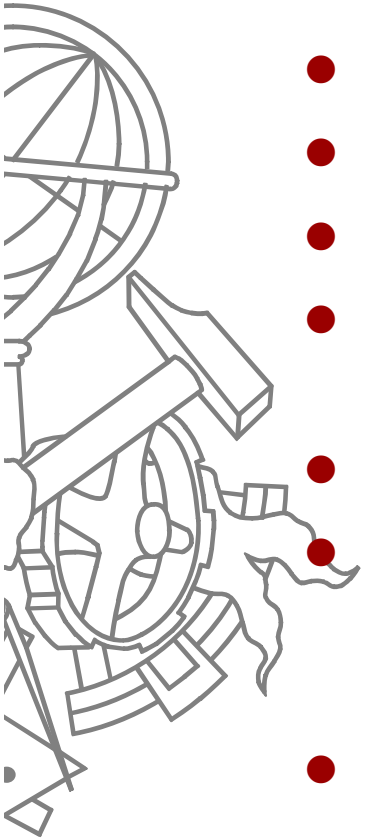
- A single instance that handles the business logic for all rows in a database table or view



fonte: Patterns of Enterprise Application Architecture

Pattern

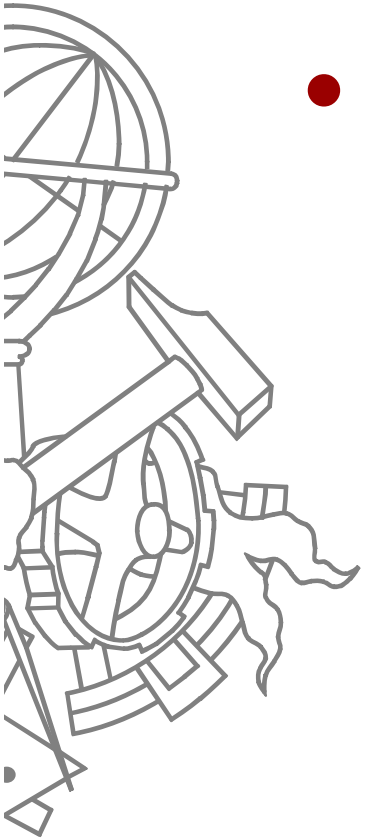
Table Module



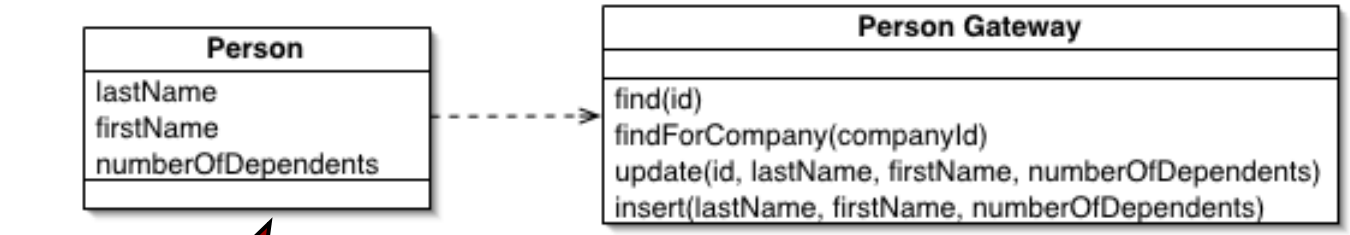
- Vários componentes (um por “tabela”)
- Métodos associados a cada “tabela”
- Uma instância para todos os registos da tabela
- Permite conceitos OO como delegação de responsabilidade
- Ideal para trabalhar com *Record Sets*
- Normalmente não guardam estado das entidades, sendo o estado passado entre camadas no *Record Set*
- Bom equilíbrio entre decomposição de código/lógica e facilidade de manutenção/evolução

Pattern

Table Data Gateway



- An object that acts as a *Gateway* to a database table. One instance handles all the rows in the table

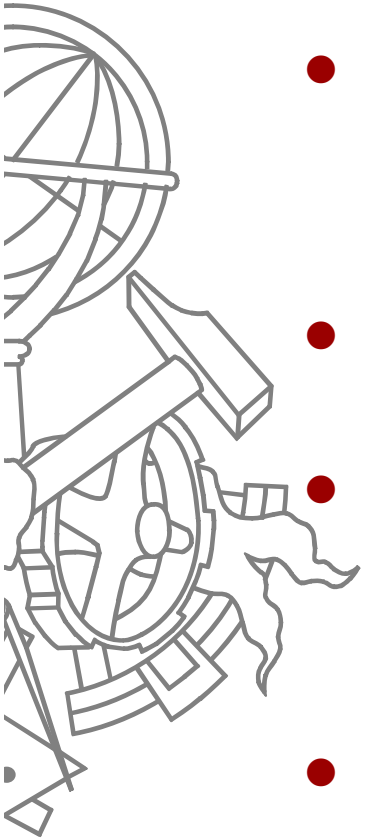


Entidade de negócio

fonte: Patterns of Enterprise Application Architecture

Pattern

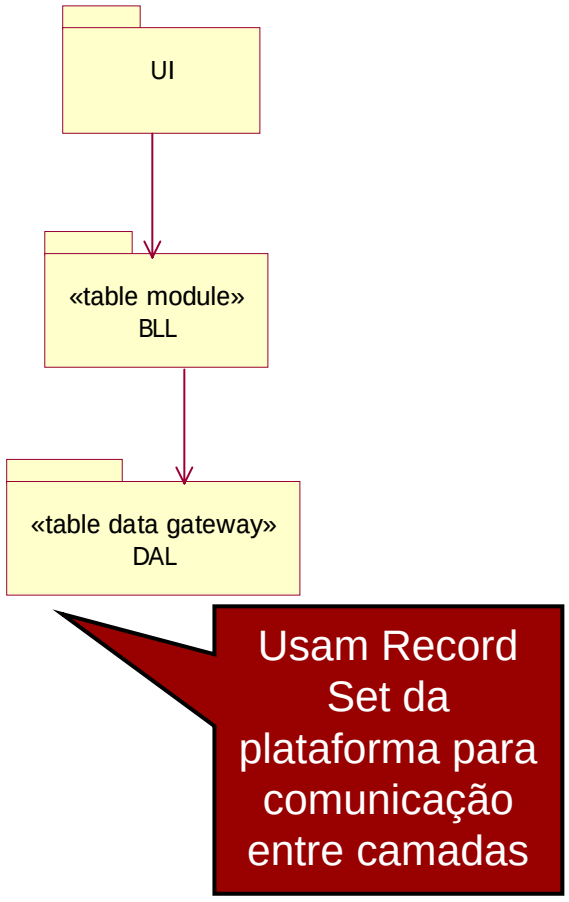
Table Data Gateway



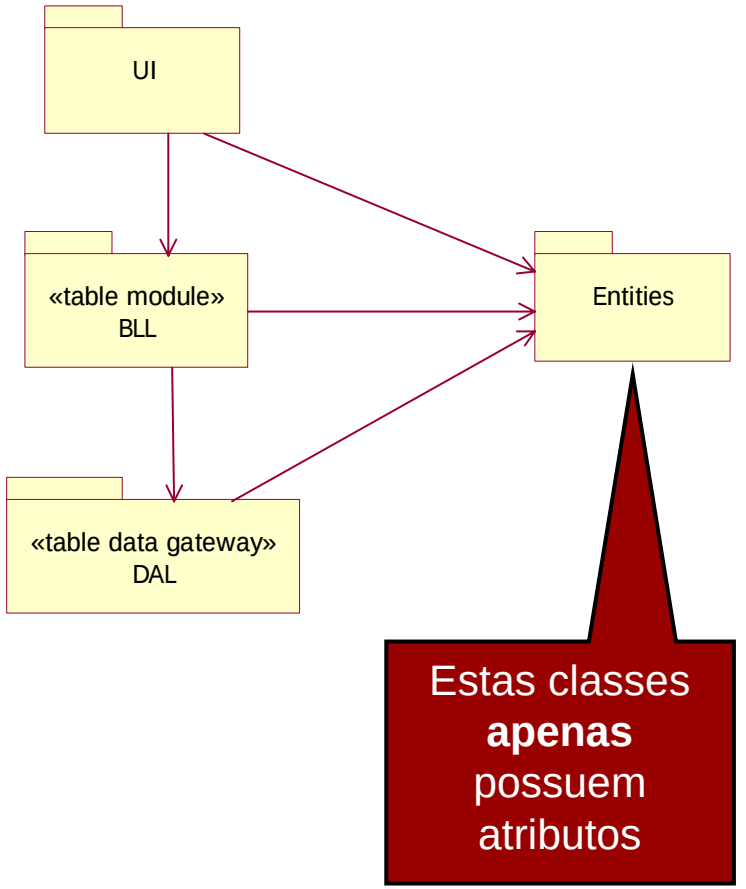
- Normalmente um para cada tabela ou vista
  - Se a plataforma suportar (ex. DataSet) pode manipular várias tabelas relacionadas (ex. cabeçalho e linha de encomendas)
- Ideal em combinação com *Table Module*
  - muitas vezes associado a *Record Set*
- Tem métodos *finder* que devolvem *RecordSets* e métodos CRUD que recebem como argumento um *RecordSet* ou os campos individuais
- Não mantém estado
- **Óptima oportunidade para usar geradores de código**

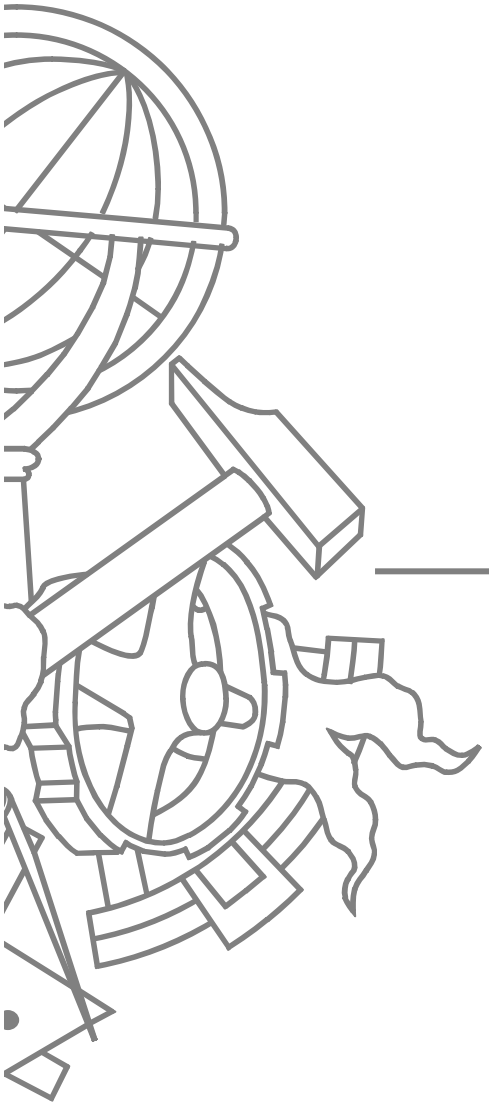
Estrutura de packages

TM+TDG



TM+TDG+CC





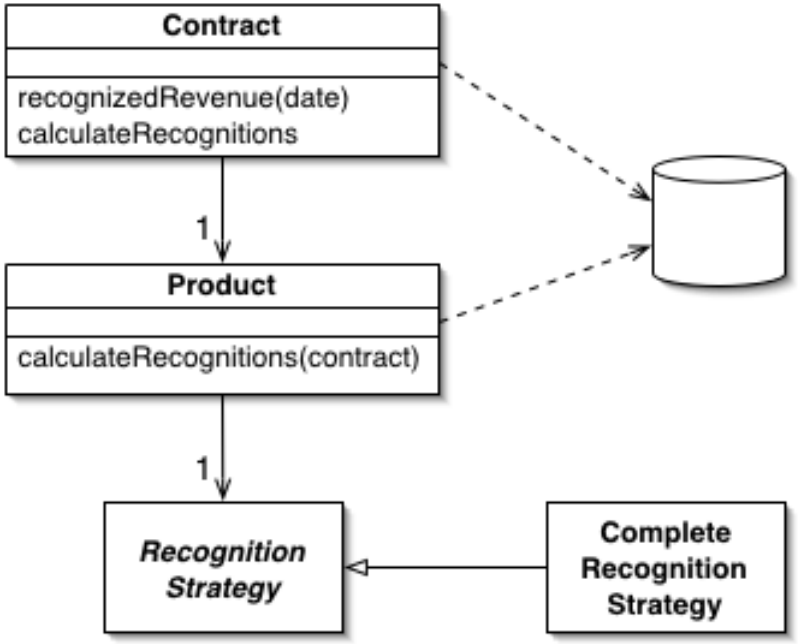
Domain Model com Active Record ou Data Mapper

Pattern

Domain Model

- An object model of the domain that incorporates both behavior and data

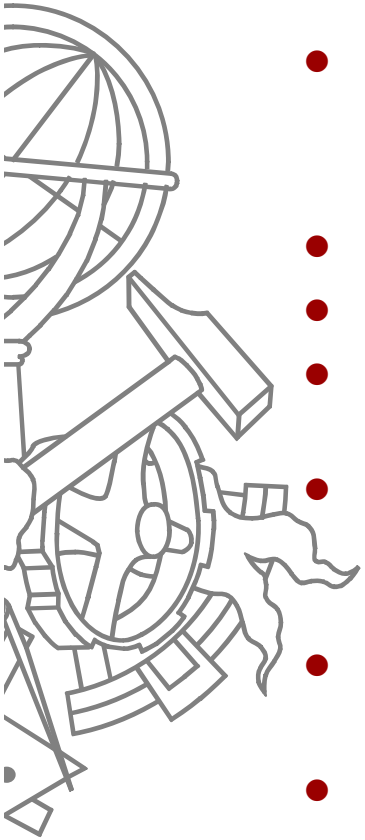
Estas classes eventualmente possuem atributos



fonte: Patterns of Enterprise Application Architecture

Pattern

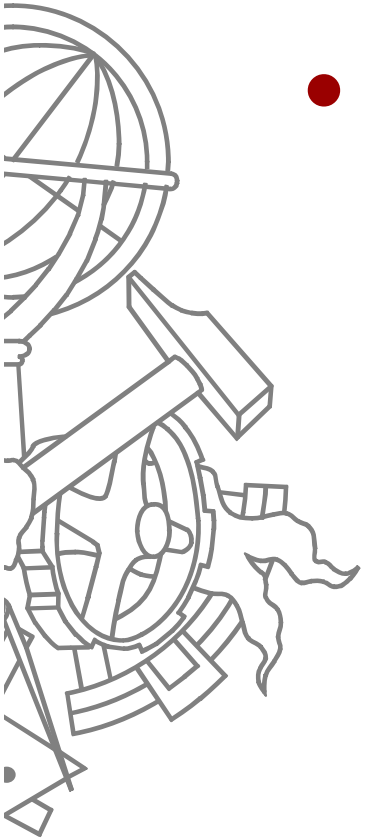
Domain Model



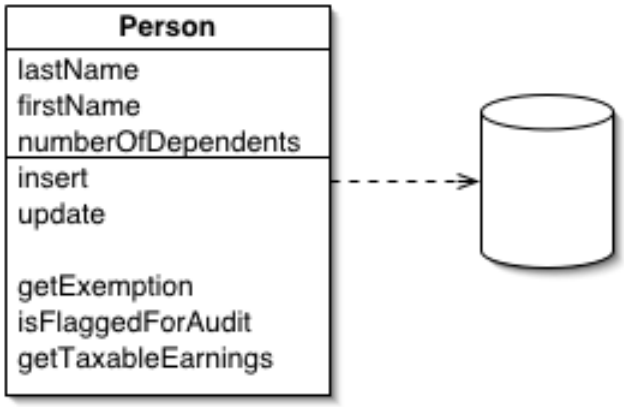
- Paradigma OO
  - Hierarquia de classes representativa do domínio do problema
  - Métodos de negócio associados a cada entidade
- Uma instância para cada registo da tabela
- Permite conceitos OO mais avançados (ex. *Strategy*)
- Pode ser demasiado complexo de implementar para situações simples
- Torna-se demasiado complexo de perceber para situações de grande dimensão
  - Curva de aprendizagem grande
- Permite a maior flexibilidade de evolução/manutenção
  - Após boa compreensão do modelo
- Dificuldades na integração com bases de dados relacionais
- Pouco eficiente para operações de grupos
  - Ex., aumentar o preço de todos os produtos em 2,4%

Pattern

Active Record



- An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data



fonte: Patterns of Enterprise Application Architecture

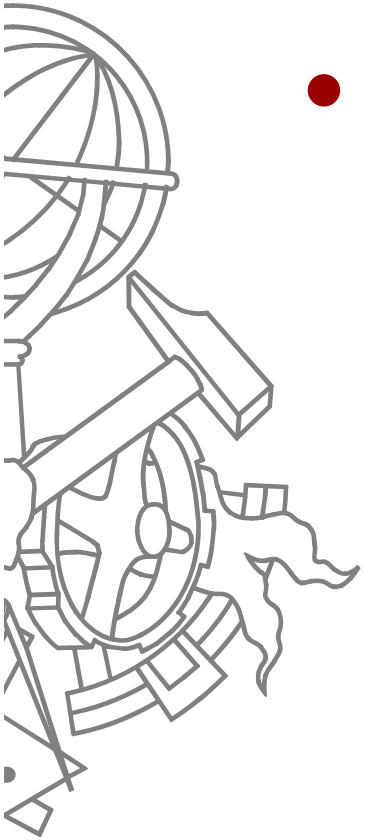
Pattern

Active Record

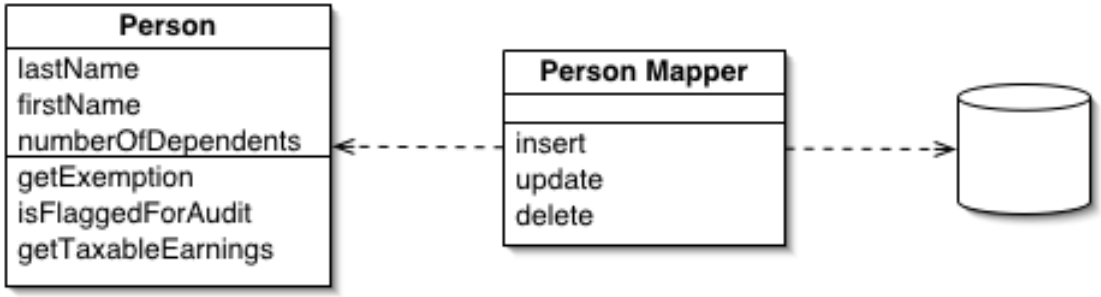
- É um *Domain Model* com classes cujos atributos mimetizam a estrutura dos registos da tabela
- Coloca o acesso a dados na lógica do domínio
  - As camadas são representadas logicamente pelo conjunto de métodos
- **Usar geradores de código sempre que possível para componente CRUD**

Pattern

Data Mapper



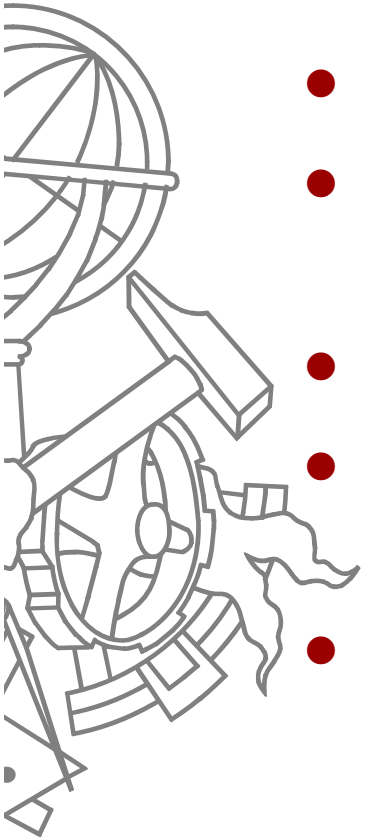
- A layer of *Mappers* that moves data between objects and a database while keeping them independent of each other and the mapper itself



fonte: Patterns of Enterprise Application Architecture

Pattern

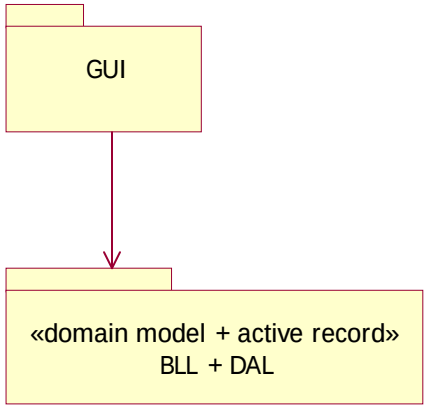
Data Mapper



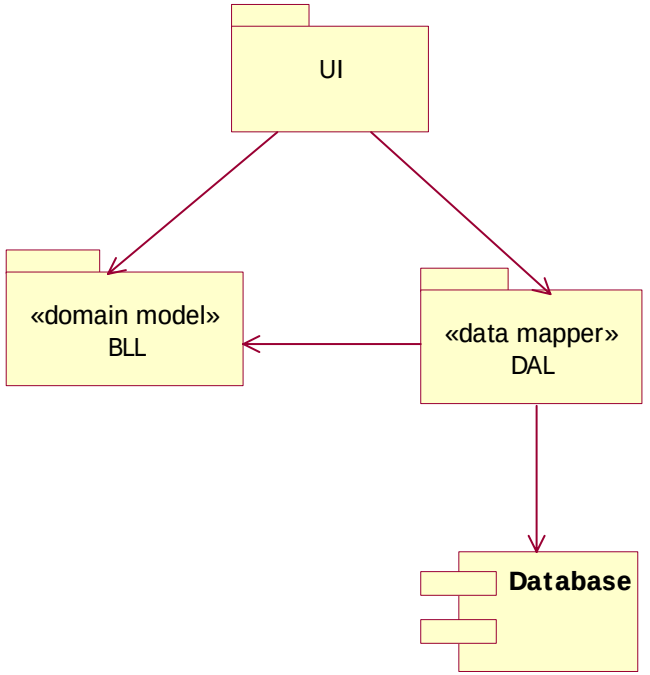
- Separar lógica do domínio da fonte de dados
- *Object Schema* e *Relational Schema* independentes
- Ideal num *Domain Model* complexo
- Permite alterações de domínio e da base de dados independentes
- **Preferencialmente, usar ferramentas de Object-Relational Mapping (ORM)**
  - **Hibernate ou nHibernate, ...**

Estrutura de Packages

DM+AR



DM+DM

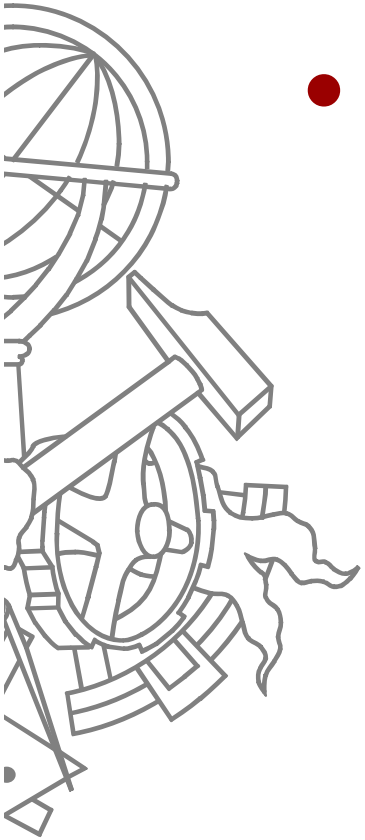


Alguns problemas adicionais com o Domain Model

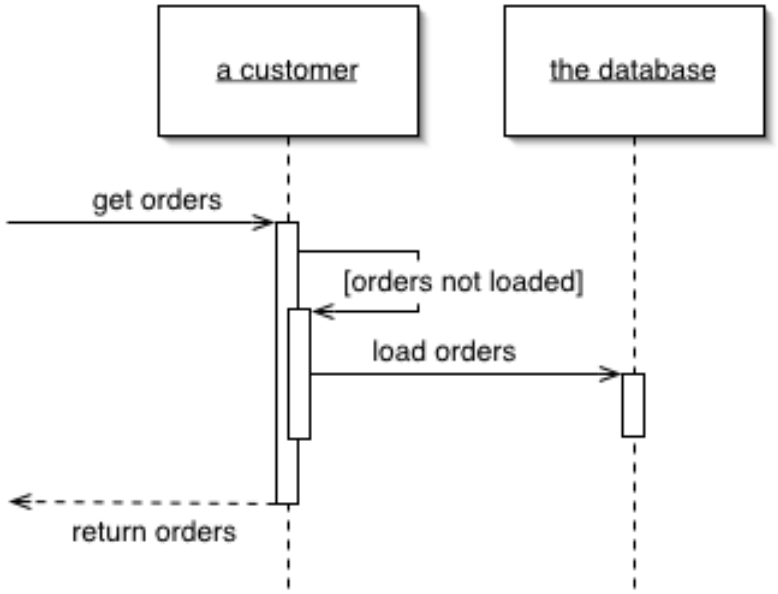
- Quando existem objectos relacionados (ex., Factura + Linhas de Factura e Produtos) como fazer relativamente aos objectos *Produto*?
  - Carregar logo para memória ou apenas quando for necessário?
- E quando mais que um objecto *Factura* se refere a um mesmo *Produto*?
  - Como garantir que não há objectos duplicados em memória?

Pattern

Lazy Load



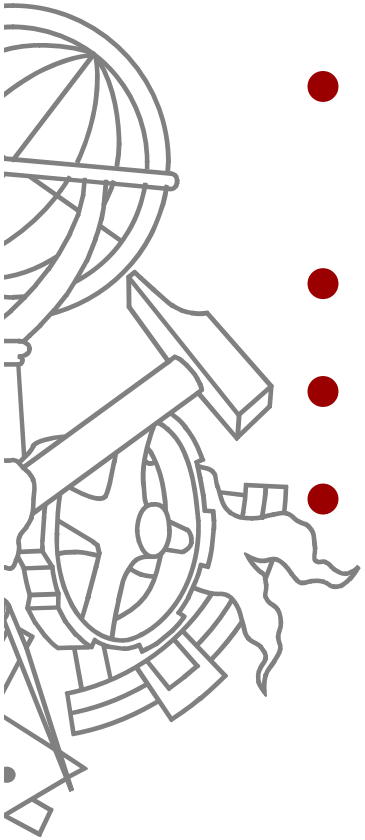
- An object that doesn't contain all of the data you need but knows how to get it.



fonte: Patterns of Enterprise Application Architecture

Pattern

Lazy Load

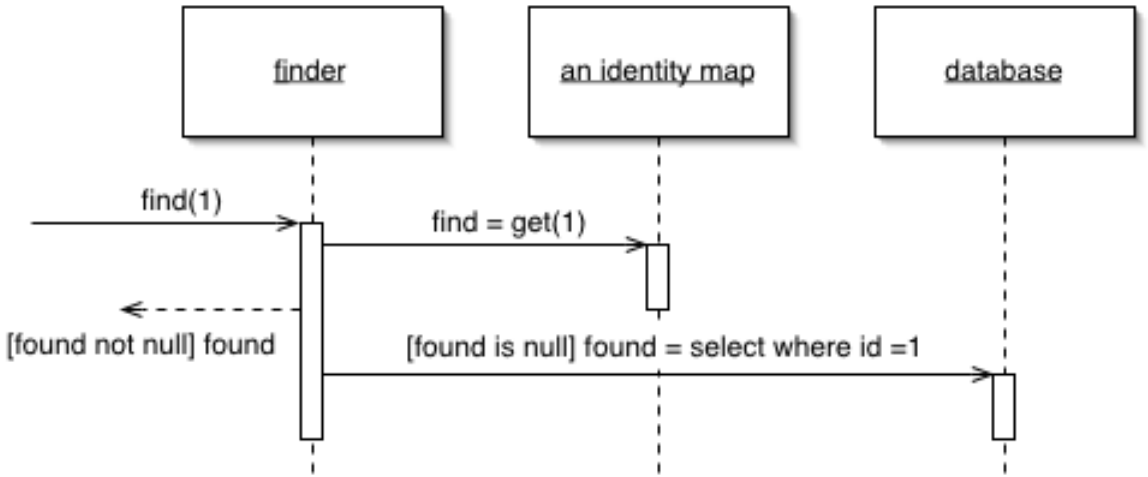


- Apenas carrega a informação quando é necessária
- Load completo / load de identidade
- Vários modos de implementação
- Pode tornar-se complexo
  - implementar apenas quando é necessário
  - **Usar geradores de código sempre que possível**

Pattern

Identity Map

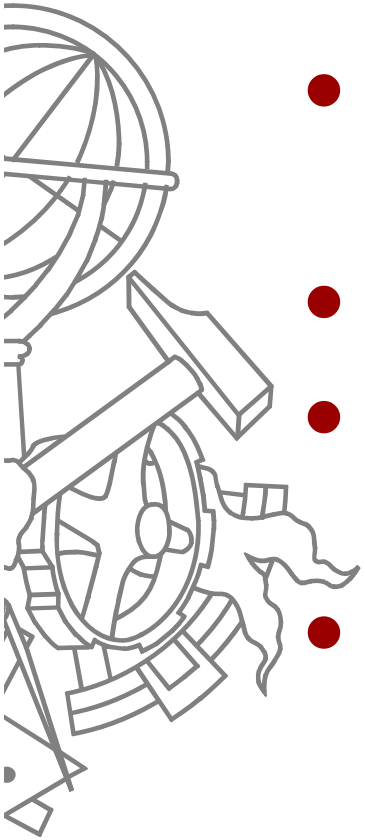
- Ensures that each object gets loaded only once by keeping every loaded object in a map. Looks up objects using the map when referring to them



fonte: Patterns of Enterprise Application Architecture

Pattern

Identity Map



- Mantém um lista de objectos lidos da base de dados
- Usado pelos métodos *finder*
- Uma lista por tabela ou sessão
  - Normalmente implementado como Singleton
- Pode funcionar como uma cache
  - Mas não deve