Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

2019/2020

Construção de Sistemas de Software

Padrões para a Camada de Acesso aos Dados
— *Active Record* e *Data Mapper* —
e
Object Relational Mapping

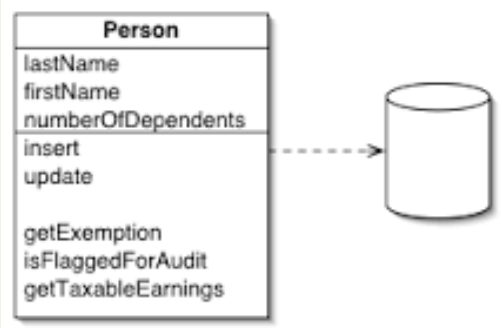
Padrões para as várias camadas

Apresentação	MVC Front controller Page controller Page template
Negócio	Domain Model Transaction Script Table Module
Dados	✓ Row data gateway ✓ Table data gateway Active Record Data Mapper

Active Record

- Os dois padrões para organizar a camada de dados que vimos anteriormente, baseados ***gateways***, eram **puros**
- O ***Active Record*** é um padrão **híbrido** já que se aplica à organização de código que trata simultaneamente da lógica de negócio e da lógica de acesso aos dados

- Um ***active record*** é um objeto que
  - embrulha uma entrada de uma tabela/vista encapsulando o acesso à base de dados
  - adiciona lógica de negócio a esses dados



Active Record

- A classe cujos objetos são registos ativos de uma tabela **T**
  - tem um **atributo** por cada **coluna** de **T**
  - tem construtores que recebem os valores dos atributos necessários para a construção e *getters/setters* para atributos
  - tem métodos de instância ***insert()***, ***delete()*** e ***update()*** para inserir um registo, apagar ou alterar o registo representado pelo *this* na tabela **T**
  - tem métodos de instância que implementam lógica de negócio
  - tem métodos de classe que são ***finders*** e que constroem *active records* a partir de *Result Sets*
  - tem métodos de classe para fazer o ***load*** a partir de um *Result Set*
- A inserção de novos registos, atualização ou eliminação de registos na base de dados é feita tal como no RDGW.
- Idem para a pesquisa.

Active Record

- Como evolução do ***Domain Model***
  - Dotar as classes do **modelo de domínio** da capacidade de **persistência de informação**
  - Tem de haver um "alinhamento" entre os conceitos do domínio e as tabelas da base de dados
  - Além dos métodos de negócio, a classe tem responsabilidade de persistir os seus objetos
- Como evolução do ***Transaction Script*** + ***RDGW***
  - Como forma de evitar repetição de código

Utilização

- É uma boa escolha enquanto os conceitos de domínio e as tabelas da BD coincidem.
  - Isto só acontece quando a lógica de negócio é pouco complexa (por exemplo, CRUD) e validações e derivações podem ser feitas localmente.
  - Quando a complexidade da lógica de negócio começa a crescer deixa de ser praticável
- **Consequências**
  - Todos os registos ativos estão ***coupled*** **com a API** de acesso aos dados (por exemplo, JDBC)
  - Há uma **dispersão de SQL**, acesso a dados, etc., por várias classes (os objetos ativos)
  - A lógica de acesso aos dados não está separada da lógica de negócio, o que **não permite a sua evolução de forma independente**

Data Mapper

- O ***Data Mapper*** é um padrão em que são usados ***mappers*** para encapsular toda a lógica de conversão de dados representados no modelo de objetos e no modelo relacional (bidirecional)

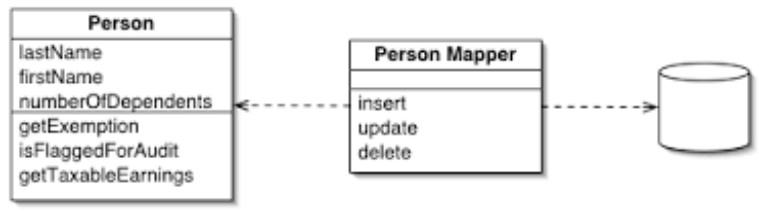
- Um ***mapper*** é um objeto que
  - move os dados em **memória** — grafos de **objetos** — para **registos** **persistidos** na base de dados
  - E vice-versa

- A adoção de *mappers* permite que cada um dos modelos (objetos e relacional) evolua de forma independente

- É a solução apropriada quando se estrutura a camada de negócio com o *Domain Model*

Data Mapper

- As classes da camada de negócio não têm de saber nada sobre como os dados estão persistidos
  - Não têm informação sobre comandos SQL
  - Não conhecem o esquema de tabelas
- As classes da camada de negócio não sabem sequer da existência da camada de mapeamento de dados



- É o padrão para a persistência de objetos em memória mais complicado de concretizar
- Fazer implementações “à medida” não é recomendável

ORM: Object-Relational Mapping

- A persistência de um **modelo de objetos em memória** numa base de dados relacional exige sempre ter um mapeamento entre este modelo e um **modelo relacional persistido**
- O ORM é afetado pelo ***Object-relational Impedance Mismatch***: o modelo de objetos e o modelo relacional são duas formas muito diferentes de **estruturar** dados
  - Referência vs. Chave primária
  - Atributos com listas de valores vs. campos 1 valor
  - Herança vs. ?
  - Ligações unidirecionais vs. relações simétricas

  que suportam duas abordagens diferentes de acesso

  - Atravessar grafos de objetos através das suas relações vs fazer *join* das linhas de dados das tabelas
- Mas há dificuldades ainda maiores relacionadas com aspetos **comportamentais**

ORM: Object-Relational Mapping

- Os dois modelos são sujeitos a mudanças e há a possibilidade de mudanças concorrentes
- À medida que se vão carregando objetos em memória é preciso ter em atenção que não se carrega **o mesmo objeto duas vezes**, pois isso permite alterações concorrentes de dois objetos que representam o mesmo registo de uma tabela.
- À medida que se vão carregando objetos e os vamos modificando, é preciso garantir que o estado da base de dados com que se trabalha permanece **consistente** e que a **leitura é isolada**, i.e., não há outros processos a modificar os objetos lidos, que contém informação que está a ser usada.
- Em modelos de objetos é comum ter objetos ligados e esperar que os **objetos ligados são carregados juntos**. Como isto pode implicar carregar para memória um enorme grafo, é preciso ter formas de reduzir o que é carregado e só carregar quando, e se, as ligações forem usadas.

ORM: Padrões

- **Estruturais**
  - *Identity Field*
  - *Foreign Key Mapping*
  - *Association Table Mapping*
  - *Dependent Mapping*
  - *Embedded Value*
  - *Single Table Inheritance*
  - *Class Table Inheritance*
  - *Concrete Table Inheritance*
- **Comportamentais**
  - *Identity Map*
  - *Unit of Work*
  - *Lazy Load*
  - *Optimistic Offline Lock*
  - *Query Object*

ORM: Object-Relational Mapping

***ORM** "is the Vietnam of Computer Science (...) represents a quagmire which starts well, gets more complicated as time passes, and before long entraps its users in a commitment that has no clear demarcation point, no clear win conditions, and no clear exit strategy."* *Ted Neward,2004*

- Existem diversas soluções para o ORM que
  - criam tudo o que é necessário para concretizar o mapeamento entre os dois modelos
  - a partir de uma descrição de alto nível do mesmo
- Assim, em vez de aprender em abstrato sobre todos os padrões que ajudam a implementar o ORM e *data mappers*, vamos ver estes padrões enquadrados no contexto de uma família de soluções ORM para Java
- As opções que estas soluções oferecem para definir o mapeamento estão fortemente relacionados com estes padrões

ORM: Object-Relational Mapping

- Exemplos de Soluções ORM para Java
  - **TopLink** (Oracle)
  - **EclipseLink** (Eclipse Foundation)
  - **Hibernate** (JBoss/RedHat)

JPA

Provides standards based Object-Relational persistence solution with additional support for many advanced features. EclipseLink JPA provides advanced support for leading relational databases and Java containers.

JPA Provider

In addition to its own "native" API, Hibernate is also an implementation of the Java Persistence API (JPA) specification. As such, it can be easily used in any environment supporting JPA including Java SE applications, Java EE application servers, Enterprise OSGi containers, etc.

JPA

- **JPA = Java Persistence API**

  API do Java EE que define a norma para mapear modelos de objetos Java (POJOs) em modelos relacionais

  - **javax.persistence**
- O mapeamento é baseado em ***metadata***
  - o processo é repetitivo e pode ser automatizado em função de *metadata* nomeadamente o esquema das tabelas e das classes
  - no caso de JPA os metadados são especificados através de anotações

Sumário

- O padrão *active record* e sua implementação. Relação deste padrão com outros estudados anteriormente.
- O padrão *data mapper.*
- O mapeamento entre o modelo de objetos e o modelo relacional (ORM) e as dificuldades subjacentes.
- Discussão sobre a implementação de soluções ORM: soluções à medida vs chave na mão.

Quiz: The Object-to-Table Mapping Problem

O/R mappings can take place in a variety of forms, the easiest of which to recognize is the automated O/R mapping tool, such as [redacted]. Another form of mapping is the hand-coded one, in which programmers use relational-oriented tools, such as [redacted], to access relational data and extract it into a form more pleasing to object-minded developers "by hand". A third is to simply accept the shape of the relational data as "the" model from which to operate, and slave the objects around it to this approach; this is also known in the patterns lexicon as [redacted] or [redacted]; many data-access layers in both Java and .NET use this approach and combine it with code-generation to simplify the development of that layer. Sometimes we build objects around the relational/table model, put some additional behavior around it, and call it [redacted]

JDBC – Hibernate – Active Record – EclipseLink – Row Data Gateway – Table Data Gateway

Quiz: Solução

The Object-to-Table Mapping Problem

O/R mappings can take place in a variety of forms, the easiest of which to recognize is the automated O/R mapping tool, such as TopLink, Hibernate / NHibernate, or Gentle.NET. Another form of mapping is the hand-coded one, in which programmers use relational-oriented tools, such as JDBC or ADO.NET, to access relational data and extract it into a form more pleasing to object-minded developers “by hand”. A third is to simply accept the shape of the relational data as “the” model from which to operate, and slave the objects around it to this approach; this is also known in the patterns lexicon as Table Data Gateway [PEAA, 144] or Row Data Gateway [PEAA 152]; many data-access layers in both Java and .NET use this approach and combine it with code-generation to simplify the development of that layer. Sometimes we build objects around the relational/table model, put some additional behavior around it, and call it Active Record [PEAA,