Semestre de Inverno 2012-2013

**Modelação e Padões de Desenho**

**DaoGenerator**

**- Instituto Superior de Engenharia de Lisboa –**

**Professor: Miguel Carvalho**

**Trabalho Realizado Por:**

**- Bruno Costa — nº 36868;**

# Introdução

No âmbito do trabalho final da unidade curricular, foi proposta a realização de uma Framework que gera automaticamente ***data access* *object* (DAO).** Durante o relatório será abordada a implementação e serão explicadas algumas decisões tomadas; apresentar-se-ão esquemas da solução; serão discutidos alguns problemas encontrados; e serão explicadas outras soluções para alguns problemas.

# DaoGenerator

## Enunciado

Seguindo o [guia](https://github.com/isel-leic-mpd/mpd-2013/wiki/Trabalho-Final) dado pelo docente pode-se encontrar o primeiro problema deste trabalho final. É necessário conseguir, em tempo de execução, criar uma classe derivada da interface que especifica as operações a serem realizadas sobre a base de dados, neste caso o ProductDao. Este problema foi explicado pelo docente, que indicou que o JDK já oferecia um mecanismo para o satisfazer. Esse mecanismo é disponibilizado através da chamada ao método Proxy.newProxyInstance. O método [Build.make](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/Builder.java) faz uso desse mecanismo.

O trabalho realizado por este proxy, excluindo outros não relevantes para a discussão, é fazer com que uma chamada a um método na interface seja equivalente à chamada do mesmo método na classe gerada em runtime. Um dos parâmetros do método é uma instância de [InvocationHandler](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/InvocationHandler.html). Esta interface especifica apenas o método invoke. Por isso é necessário fazer com que [neste método](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/DAOInvocationHandler.java), cada uma das chamas à interface seja correspondida uma operação à base de dados. É necessário também ter em conta o desempenho do mecanismo e torná-lo o mais eficiente possível recorrendo a cache.

## Terminologia

@CRUD – anotações Create, Read, Update e Delete  
*daoHandler* – instância de *DaoInvocationHandler*

## InvocationHandler

Vou falar então na implementação do daoHandler e as mudanças que foi sofrendo ao longo do tempo. No início, o código de uma operação completa à base de dados estava no método invoke, para ter uma maior percepção de acções necessárias. Depois pus em prática os conhecimentos adquiridos na unidade curricular, aplicando padrões de desenho. Criei o package daoGenerator.g14.structure.funcs onde estão todas as DaoAbstranctFunc. Posteriormente, falarei mais sobre este package, das suas classes, e nos padrões de desenho que foram aplicados.

[Numa primeira abordagem](https://github.com/isel-leic-mpd/LI41D-G14/blob/11441e70a6d781f19e2c610ce91137f68533034d/daoGenerator/src/main/java/daoGenerator/g14/strucutre/DAOInvocationHandler.java), fiz o mapeamento de cada uma das @CRUD na criação duma instância concreta de DaoAbstractFunc. Não há distinção da leitura de um elemento da leitura de vários elementos. Ambas as operações eram mapeadas no mesmo tipo de objecto ([DaoReadFunc](https://github.com/isel-leic-mpd/LI41D-G14/blob/888a2fbffba28c9d6054483e672476bdf2d5d3b2/daoGenerator/src/main/java/daoGenerator/g14/strucutre/DAOReadFunc.java)).

[Numa segunda abordagem](https://github.com/isel-leic-mpd/LI41D-G14/blob/6145a2c256386014e7e54ea8c2a3e54b2343ecd8/daoGenerator/src/main/java/daoGenerator/g14/strucutre/DAOInvocationHandler.java), usei uma implementação, por mim modificada, do padrão Visitor. Modificada, porque as classes de domínio, neste caso as @CRUD, não fazem uma segunda chamada ao Visitor (o daoHandler). Segui esta abordagem para não ter que guardar uma variável instância de DaoAbstractFunc no daoHandler.

Finalmente, acrescentei suporte para diferenciar a leitura de um elemento da leitura de vários elementos. Esta alteração teve mais impacto na forma como a anotação Read estava implementada.

Na minha solução, as @CRUD servem apenas para efeitos de *metadata* e não têm mais nenhum código para além do método chamado pelo daoHandler. Este método é definido na interface [DaoVisitable](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/cmdTypes/DaoVisitable.java) que todas as @CRUD implementam. Uma outra solução seria elas próprias serem DaoAbstractFunc e terem todo o código para interagir com a base de dados.

Neste diagrama UML podem ser vistas as dependências (incluindo herança e associações) entre as classes que descrevi. Será explicado o papel de DaoAbstractFactory adiante.

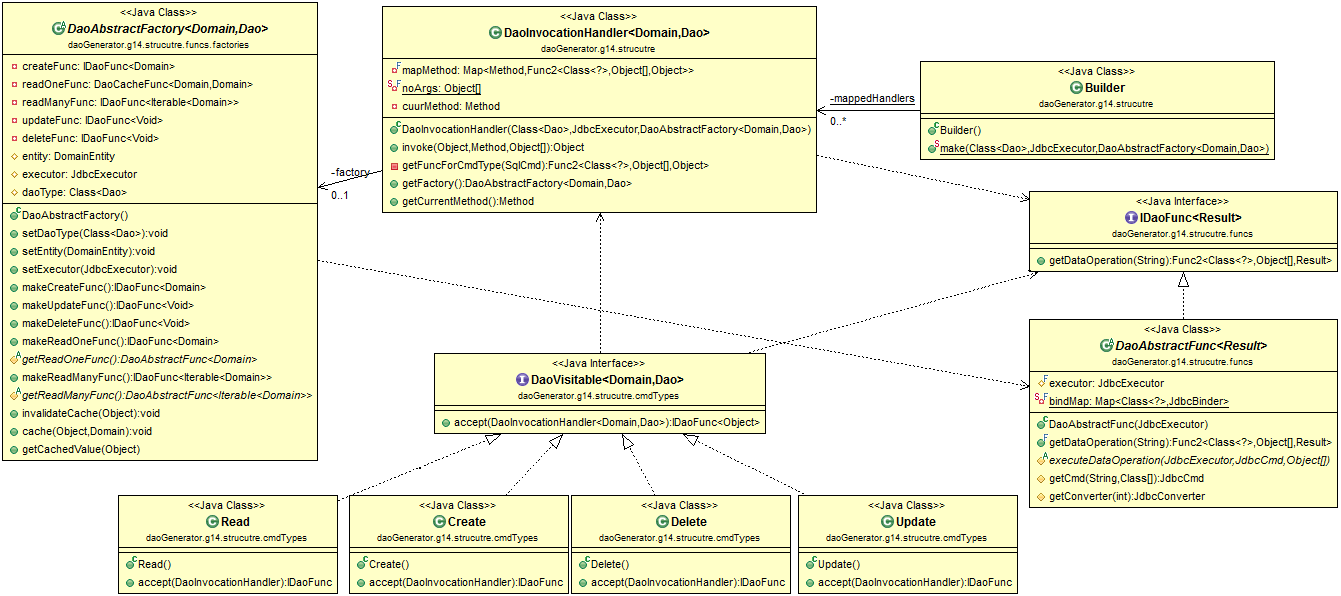


Figure - Diagrama UML dos componentes centrais da Framework

## Package daoGenerator.g14.structure.funcs

Falarei agora sobre a implementação das classes deste package, que como já mencionei, a sua responsabilidade é de fazer a operação à base de dados.

Eu tentei reduzir ao máximo o trabalho necessário de cada uma das operações, sendo somente necessário a operação sobre a base de dados especificada no método executeDataOperation. Para a leitura e criação ainda é necessário o método getConverter, que converte um resultset numa instância de domínio. E assim foi aplicado o padrão *Template Method.*

A classe [DaoAbstractFunc](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/funcs/DaoAbstractFunc.java) define o método esqueleto getDataOperation que a partir da query e dos parâmetros passados, determina o comando e chama o método executeDataOperation (da classe especifica).

### Caching

Foi também aplicado o padrão Decorator. Este padrão surgiu como solução para resolver o problema de fazer cache às chamadas de getById. Para não quebrar a lógica de DaoAbstractFunc, foi adicionada essa funcionalidade na classe [DaoCacheFunc](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/funcs/DaoCacheFunc.java).

Para acrescentar caching, segundo a minha solução, é necessário redefinir o método getDataOperation. O problema é que redefinir ou reutilizar o método getDataOperation de DaoAbstractFunc é complicado, uma vez que o código está numa classe anónima a que chamei Func (Func2). Devido à simplicidade do algoritmo resolvi meter o método getDataOperation numa interface ([IDaoFunc](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/funcs/IDaoFunc.java)) e implementar esta interface em DaoAbstractFunc e DaoCacheFunc. Não pensei em outras alternativas para resolver o problema de caching.

Para determinar se uma dada função pode ou não fazer cache também se podia recorrer à definição de um método que indicasse isso.

Ainda relacionado à cache; é necessário entender como a operação de Delete invalida a cache. Para a operação de delete invalidar a cache a instância de DaoDeleteFunc tem que ter conhecimento da instância de DaoReadOneFunc. Esse conhecimento foi adicionado na DaoAbstractFactory, que também passou a ter responsabilidade de criar instâncias das funções.

Eu também não considerei o Update como sendo uma operação que pode utilizar a cache, uma vez que a operação sobre a base de dados deve ser feita. Contudo o facto das instâncias de domínio serem criadas via reflexão pode ser um bom motivo para recorrer a cache.

### Relações 0..1-1

O requisito opcional de acrescentar relações entre os elementos, veio na minha solução, acrescentar um nível de complexidade de compreensão ao nível das funções. A *class* [DaoExtendedReadFunc](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/funcs/DaoExtendedReadFunc.java) necessita de uma forma diferente de encontrar o comando, uma vez que a *string sql* não é a mesma que é passado como parâmetro (tem mais colunas com as chaves estrangeiras de cada uma das relações). Foi, por isso, redefinido o método *getCmd*.

É necessário indicar as chaves estrangeiras e que relações se quer apresentar e optei por incluir isso nas classes de domínio com a anotação [DaoHolder](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/DaoHolder.java). Esta anotação indica o nome da chave na base de dados e o DAO que deve ser usado para buscar a relação à base de dados. É usado o padrão *ValueHolder*, sendo por isso um carregamento *lazy*.

Por isso a classe *DaoExtendedReadFunc* tem que ter a responsabilidade adicional de interpretar a *metadata* da classe do domínio, do DAO da classe de domínio e do DAO da classe da relação.

Apresento o diagrama UML das várias funções. Dividem-se em 3 grupos: A DaoCacheFunc, As DaoExtendedReadFunc e as DaoAbstractFunc.

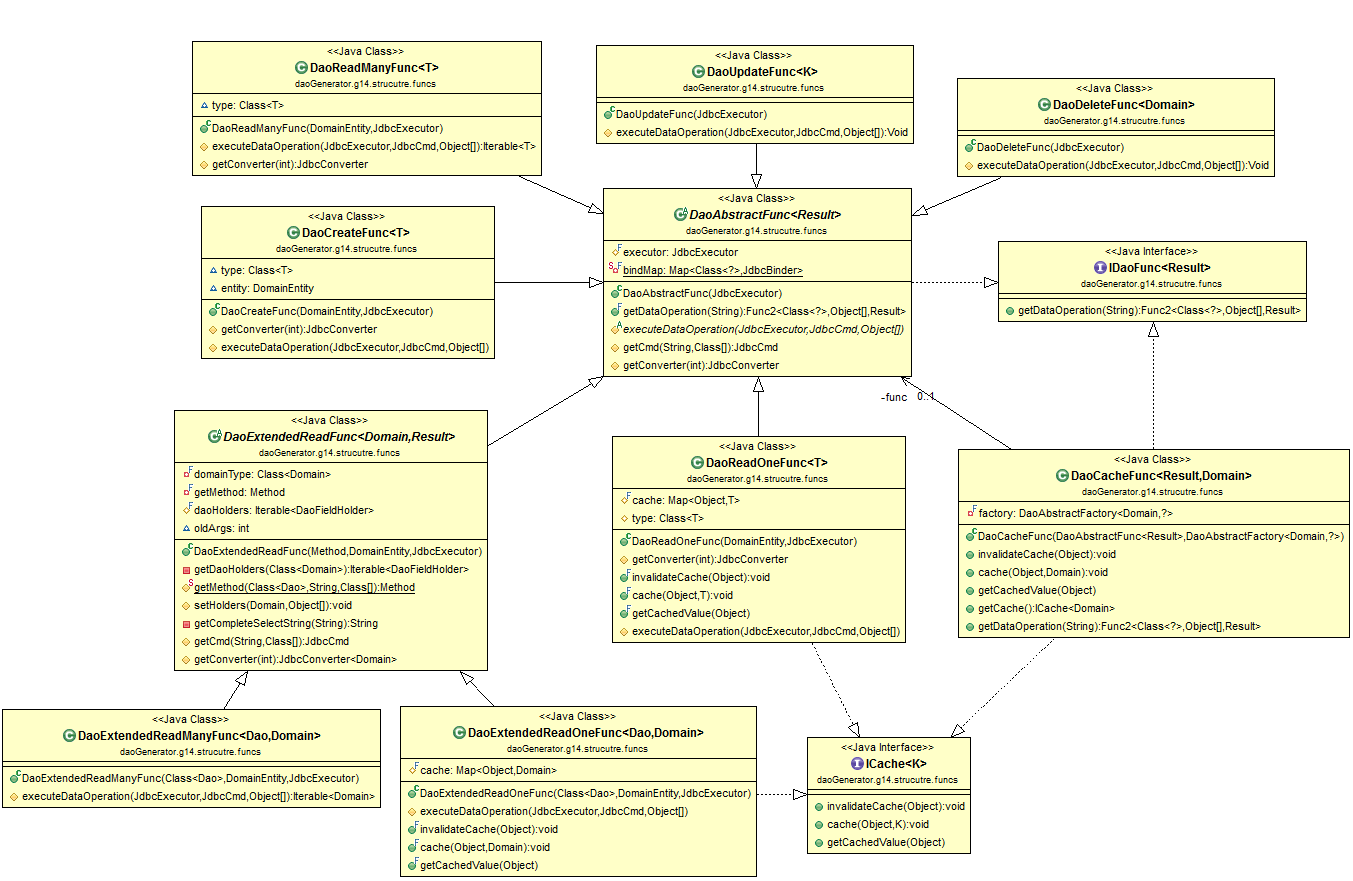


Figure - Diagrama UML das funções DAO

## Factories

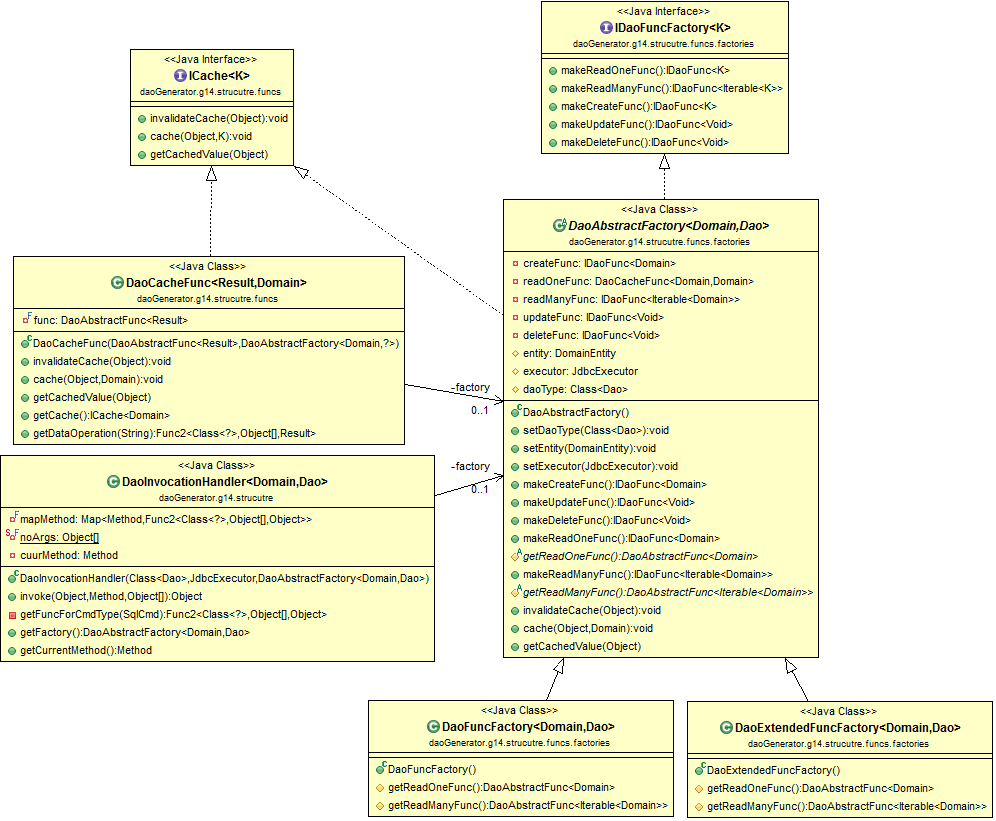
Como já expliquei as *factories* surgiram devido à necessidade de conhecimento entre as várias funções. Existem duas *factories*: a [DaoFuncFactory](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/funcs/factories/DaoFuncFactory.java) e a [DaoExtendedFuncFactory](https://github.com/isel-leic-mpd/LI41D-G14/blob/master/daoGenerator/src/main/java/daoGenerator/g14/strucutre/funcs/factories/DaoExtendedFuncFactory.java). A DaoExtendedFuncFactory respeita o requisito adicional de ter relações 1-1 usando DaoExtendedReadFunc como instâncias. O método Build.make foi alterado para que recebesse uma instância de *DaoAbstractFactory*. Apresento o diagrama UML das *factories* e o seu relacionamento com o daoHandler e *DaoCacheFunc*.

Figure - Diagrama Uml das factories

## Quick WalkThrough

**Builder**

Make – cria uma instância de DaoInvocationHandler através de Proxy.newInstance

**DaoInvocationHandler**

Invoke – faz o mapeamento entre @CRUD e chamadas a instâncias de DaoAbstractFunc

**DaoAbstractFunc**

getDataOperation - define o algoritmo para obter o comando a partir da query e argumentos

executeDataOperation – operação especifica À base de dados

**DaoCacheFunc**

getDataOperation – acrescenta a funcionalidade de cache às funções

**DaoExtendedReadFunc**

getCmd – define a nova forma de obter o comando para obter todos os campos necessários para preencher as relações na entidade de domínio

**DaoAbstractFactory**

makeXXXFunc – cria a função de determinado tipo.

Sequência de métodos (principais) chamados numa chamada a *getByID*. Alguns métodos de instância representados como estáticos.

Builder.make  
DaoInvocationHandler.Invoke (h)  
Read.accept  
h.getFactory.makeReadOneFunc (daoFunc)   
DaoReadOne ou DaoExtendedReadOneFunc.getDataOperation (func)  
func.call   
daoFunc.getCmd  
func.getConverter  
func.executeDataOperation