**Ambiente Virtuais de Execução**

**LI42D – José Simão**

**Relatório da Primeira Série de Exercícios**

****

**Carlos Pereira nº41375**

**João Gameiro nº41893**

**Paulo Alexandre nº40639**

**Objetivo**

Esta série de exercícios tem como objetivo implementar uma biblioteca ***MapperReflect*** que permite realizar o mapeamento entre membros (campos, propriedades, etc) de objetos de tipos diferentes (tipos **valor** ou **referência**).

**Realização**

Para proporcionar o mapeamento dos membros, implementámos as classes ***Mapper***, ***Mapping*** e a interface ***IMapper*** (***Mapper*** implementa ***IMapper***).

***Mapper***

Esta classe tem como responsabilidade decidir qual o tipo de mapeamento a ser realizado. No seu construtor, recebe como parâmetros dois objetos do tipo ***Type*** que representam a fonte e o destino do mapeamento, este também verifica se o construtor do tipo destino possui ou não parâmetros. Caso possua, o mapeamento é feito entre os membros do objeto fonte e os parâmetros do tipo destino. Como o mapeamento também é aplicável a tipos valor, o construtor desta classe verifica também se estamos a mapear **tipos valor** ou **tipos referência.** Caso o tipo a mapear seja do ***tipo valor***, a classe ***PrimitiveHandler*** será responsável pelo seu mapeamento.

Por omissão, o mapeamento é feito com base nas propriedades. O método ***Bind***  permite que haja mapeamento a partir dos campos ou a partir dos membros anotados com um ***custom attribute***.

O método ***Match*** é responsável por realizar a correspondência dos membros cujos nomes são passados como parâmetros. Por fim, os métodos ***Map*** (existem dois métodos com este nome, um mapeia apenas um objeto, o outro mapeia um ***array*** de objetos) são responsáveis por realizar o mapeamento consoante o objeto passado parâmetro. O método ***Map*** sempre que realiza um mapeamento, antes verifica se já há uma instância do elemento a ser mapeado (***cache***), caso não haja uma instância em cache, este método faz uma nova e guarda-a na cache.

***Mapping***

A classe ***Mapping*** representa os tipos de mapeamento que se pode realizar.

Os tipos de mapeamento são:

1. ***Fields***
2. ***Properties*** (por omissão)

De notar que a solução deve ser extensível a novas estratégias de mapeamento, posto isto foi adicionada mais uma estratégia de mapeamento.

1. ***Custom Attribute***

Para realizar a correspondência dos membros a mapear, começámos por implementar uma classe abstrata, ***Handler***, de onde irão estender as classes que tratam da correspondência de cada tipo de mapeamento.

1. ***PropertyHandler***
2. ***FieldHandler***
3. ***ParameterHandler***
4. ***AttributeHandler***
5. ***PrimitiveHandler***

***PropertyHandler*** e ***FieldHandler***

Estas duas classes tratam da correspondência por propriedades e por campos, ambas utilizam o mesmo processo de correspondência.

O construtor inicializa uma lista de ***KeyValuePair***, onde a ***key*** representa o objeto fonte e o ***value*** o tipo destino, e verifica se existem membros compatíveis, em caso afirmativo são adicionados à lista. É inicializado também um dicionário de ***KeyValuePair*** e de ***IMapper***, este dicionário é utilizado para o caso de os membros a corresponder sejam incompatíveis e de tipos complexos, esta verificação é feita no método ***LinkMembers*** (para este caso é necessário criar uma nova instância de ***IMapper***).

***ParameterHandler***

Este ***Handler*** é responsável pelo mapeamento quando na classe a mapear não existe nenhum construtor sem parâmetros, caso isto se verifique a classe é mapeada pelo construtor com parâmetros. No seu construtor, é inicializada uma lista de ***KeyValuePair***, sendo que a ***key*** um ***MemberInfo*** (pode variar entre ***FieldInfo*** ou ***PropertyInfo***) e o ***value*** um ***ParameterInfo***.

O método ***ExtractValueFromMember*** tem como objetivo verificar se o mapeamento é pelas propriedades ou pelos campos.

***AttributeHandler***

Esta classe mapeia os membros que estejam anotados com um ***custom attribute.***

O seu construtor percorrerá todos os membros presentes no tipo fonte, e verifica quais os membros que estão anotados por um ***custom attribute*** e, de seguida adiciona à lista de ***KeyValuePair*** onde a ***key*** e o ***value*** são do tipo ***MemberInfo***.

O método ***Copy*** verifica se os pares da lista são do tipo ***PropertyInfo*** ou ***FieldInfo***, estabelece os seus valores.