Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Ambientes Virtuais de Execução – Semestre de Verão – 2017/2018

Segundo Trabalho

Objectivos: Análise e manipulação programática de código intermédio com API de `System.Reflection.Emit`.
Data limite de entrega: **2 de Maio de 2018**.

No seguimento do trabalho desenvolvido em `AutoSql` pretende-se desenvolver uma nova classe `EmitDataMapper` responsável por criar uma implementação de um `IDataMapper` para uma determinada entidade de domínio (e.g. `Supplier`, `Employee`, etc) usando emissão de código IL em tempo de execução.

O objectivo é que algumas das operações realizadas via `Reflection` tais como ler ou escrever propriedades de entidades de domínio passem a ser realizadas directamente com base em código IL emitido em tempo de execução através da API de `System.Reflection.Emit`.

Neste contexto, foram realizadas algumas alterações à base de código do projecto `AutoSql` disponível em https://github.com/isel-leic-ave/autosql para que sejam visíveis os ganhos de desempenho resultantes da substituição do uso de reflexão pela utilização de código IL na leitura e escrita das propriedades. Nomeadamente foi introduzida uma *cache* com `DataSet` em `AbstractDataMapper` para que as medições de desempenho não contabilizem o tempo de leitura da base de dados.

Assim, os primeiros passos para a realização deste trabalho são:
1. introduzir no seu projecto as mesmas alterações que foram realizadas em Merge branch 'trab2' .
2. adicionar suporte para chaves primárias que NÃO sejam auto-incrementadas, a exemplo da classe `Customer` adicionada aos testes. Note que a chave primária da entidade `Customer` não é auto-incrementada. Deverá estender a funcionalidade das classes implementadas e dos mappers para suportar este tipo de chaves.

Posteriormente passe à concepção de `EmitDataMapper` cujo método `Build` retorna uma instância de uma nova classe gerada dinamicamente que estende de `DynamicDataMapper`.
Para cada entidade de domínio é gerada uma classe diferente.

Complete a implementação de `EmitDataMapper` seguindo a abordagem proposta:

1. Implemente no projecto `SqlReflectTest` uma nova classe `CustomerDataMapper` que estende de `DynamicDataMapper` e implementa os seus métodos abstractos de forma *hard coded* para a entidade `Customer`. Adicione um teste unitário para `CustomerDataMapper`.

2. Implemente `EmitDataMapper` de modo a que retorne uma implementação de uma classe semelhante à que escreveu manualmente para `CustomerDataMapper`, para a entidade especificada no parâmetro `klass` do método `Build()`.

3. Corra o teste unitário que implementou no ponto 1 para o *data mapper* obtido através de `EmitDataMapper` para a entidade `Customer`.

4. Implemente uma aplicação consola para comparar o desempenho entre os *data mappers* do ponto 1, 2 e um `new ReflectDataMapper(typeof(Customer), NORTHWIND)` usando a classe `NBench` desenvolvida nas aulas. Registe e comente os desempenhos obtidos entre as 3 abordagens.

Repita os passos anteriores para a entidade `Employee` (que é uma `struct`) e depois para a entidade `Product`. Note que no caso da entidade `Product` o seu *data mapper* depende dos *data mappers* de `Supplier` e `Category` que neste caso deverão ser construídos usando o `EmitDataMapper` e NÃO `ReflectDataMapper`.