Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Ambientes Virtuais de Execução – Semestre de Verão – 2020/2021

Trabalho de Época Especial

Data limite de entrega: 8 de Setembro de 2021

Pretende-se implementar uma *framework* que disponibiliza um método que produz uma nova instância de uma classe de domínio cujas propriedades e métodos virtuais são acrescentados com **funcionalidades adicionais**.
O exemplo seguinte apresenta um caso de criação de um objecto de domínio `Stock`. Os parâmetros de `Build` serão passados como argumentos ao construtor de `Stock` na sua instanciação.

```
Stock st = Enhancer.Build<Stock>("Apple", "Dow Jones");
```

As **funcionalidades adicionais** são especificadas por *custom attributes* anotados sobre as propriedades e métodos virtuais, conforme o **exemplo** seguinte. Em comentário é indicado aquilo que se pretende que as propriedades ou métodos anotados passem a verificar no objecto retornado pela função `Build` de `Enhancer`.

```
class Stock
{
    public Stock(string name, string index) { ... }

    [NonNull]
    public virtual string Market { get; set; } // set dará excepção para valores null
    [Min(73)]
    public virtual long Quote { get; set; } // set dará excepção para valores < 73
    [Min(0.325)]
    public virtual double Rate { get; set; } // set dará excepção para valores < 0,325
    [Accept("Jenny", "Lily", "Valery")]
    public virtual string Trader{get; set; } // set só aceita valores Jenny, Lily e Valery
    [Max(58)]
    public virtual int Price { get; set; } // set dará excepção para valores > 58

    // dará excepção se o estado de this ou algum dos parâmetros tiver sido alterado
    // pela execução do método anotado -- BuildInterest
    [NoEffects]
    public double BuildInterest(Portfolio port, Store st) { ... }
}
```

Estratégia de Implementação:

- O projecto `Enhancer` deve tirar partido da API de reflexão e manipulação programática de código intermédio através de `System.Reflection.Emit`.
- A solução **NÃO pode estar comprometida** com os exemplos dados (e.g. `Min`, `NonNull`, etc), podendo ser extensível a outros casos a adicionar no futuro sem necessidade de alterar `Enhancer`.
- A função `Build<T>(params object[] args)` deve retornar uma nova instância de uma nova classe derivada de `T`, seja `T'`, que redefine os métodos virtuais de `T`. A nova classe `T'` é criada dinamicamente com recurso à API de `System.Reflection.Emit`.
- Por cada método `M` definido em `T` que esteja anotado, deve ser criado uma redefinição desse método `M'` na classe `T'`.
- O método `M'` deve chamar o método `M` da base e o método `Check` do *custom attribute* anotado em `M`.
- Todos os *custom attributes* do exemplo devem ter um tipo base comum – `EnhanceAttribute` – com um método abstracto `Check(object[] args)` que recebe os argumentos do método anotado.

Entregar:

- Solução Visual Studio com 3 projectos: `Enhancer`, Testes Unitários e Modelo de Domínio usado pelos testes unitários. Os testes unitários devem verificar todas as situações de sucesso e insucesso dos *custom attributes* anteriores.
- **Relatório** com explicação da arquitectura da solução implementada em `Enhancer` e todos os detalhes necessários à compreensão.