Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Ambientes Virtuais de Execução
2016

A biblioteca **settler** permite popular aleatoriamente os campos ou propriedades de objectos de domínio. O exemplo seguinte apresenta um caso de construção de um objecto `Fixture<Student>` a partir do qual é possível criar instâncias de Student, cujas as propriedades ou campos são preenchidos com valores aleatórios.

```
Fixture<Student> fix = AutoFixture.For<Student>();
Student s1 = fix.New();
Student s2 = fix.New();
```

O método `New` é recursivo e sempre que um membro NÃO seja de tipo `String`, ou primitivo, deve gerar uma instância do tipo desse membro e popular essa instância automaticamente.

Se um membro for de tipo *array* então deve ser instanciado um *array* com um tamanho aleatório e populado com elementos do seu tipo.

Uma instância `Fixture` pode opcionalmente ser configurável através dos métodos : `Member()` e `Singleton()`. No exemplo seguintes, o método `Member()` recebe por parâmetro o conjunto de valores possível de ser atribuído à respectiva propriedade ou campo.

```
Fixture<Student> fix = AutoFixture
   .For<Student>()
   .Member("Name", "Jose ", "Maria Papoila", "Augusto Seabra") // Field or property with the name Name
   .Member("Nr", 8713, 2312, 23123, 131, 54534);               // Field or property with the name Nr
```

Uma instância `Fixture` pode ainda ser configurável com outra Fixture referindo-se a um tipo de campo ou propriedade

```
Fixture<School> fixSchool = AutoFixture
   .For<School>()
   .Member("Name", "ISEL")
   .Singleton();                              // Method New() must return always the same object.
Fixture<Student> fix = AutoFixture
   .For<Student>()
   .Member("School", fixSchool); // All Students will point to the same School object.
```

A classe `SettlerEmit` tem uma API semelhante à de `Settler` mas uma implementação distinta que melhore o seu desempenho.

O objectivo é que algumas das operações realizadas via `Reflection` tais como: afectar propriedades ou instanciar uma Entidade de Domínio (e.g. `Student`, `School`, etc) sejam realizadas directamente com base em código IL emitido em tempo de execução.

O `SettlerEmit` suporta a instanciação de classes com, ou sem, construtor sem parâmetros.
No primeiro caso a instância é iniciada através das suas propriedades com valores aleatórios.
No segundo caso, são passados valores aleatórios aos parâmetros do construtor seleccionado.
Ambas as abordagens são concretizadas através de implementações geradas dinamicamente para a Entidade de Domínio em questão.

Por omissão uma instância de `Fixture<T>` para uma determinada entidade de domínio `T` inicializa automaticamente com valores aleatórios ou todos os campos (ou propriedades) da nova instância no caso de existir um construtor sem parâmetros; ou todos os argumentos recebidos pelo construtor.

Há possibilidade de indicar quais os campos (ou propriedades) ou argumentos do construtor que devem ser ignorados por uma instância de `Fixture<T>`. Para tal o utilizador pode especificar:

- o nome do campo (ou propriedade) ou argumento que não deve ser iniciado conforme exemplo da Figura 1.
- o tipo do *custtom attribute* anotado no campo (ou propriedade) ou argumento que devem ser ignorados, conforme exemplo da Figura 2.

```
Fixture<Student> fix = AutoFixture
  .For<Student>()
  .Ignore("Name");
Student s = fix.New(); // Não afecta a propriedade Name
```

Figura 1

```
Fixture<Student> fix = AutoFixture
  .For<Student>()
  .Ignore<NonFixtureAttribute>();
// Não afecta propriedades anotadas com NonFixture
Student s = fix.New();
```

Figura 2

No caso de ignorado um argumento do construtor então deve ser passado o valor *default* em vez de aleatório.

Uma instância de `Fixture<T>` pode ser configurada com uma função `Func<R>` que deve fornecer os valores a usar na inicialização de um determinado campo (ou propriedade) ou argumento do construtor.
No exemplo seguinte é especificado que a propriedade `BirthDate` de `Student` deve ser iniciada com os valores gerados pela função passada como parâmetro ao método `Member`.

```
Random rand = new Random();
DateTime dt = new DateTime(1970, 1, 1);
Fixture<Student> fix = AutoFixture
  .For<Student>()
  .Member("BirthDate", () => dt.AddMonths(rand.Next(600)));
```

Figura 3

O método `Member` verifica se a o tipo do campo (ou propriedade) ou argumento especificado é compatível com o tipo de retorno da função. No exemplo da Figura 3 verifica se `BirthDate` é compatível com `DateTime`. Caso não seja lança excepção.

O método `Member` também aceita a função para propriedades que sejam do tipo `[]R` ou `IEnumerable<R>` em que `R` seja compatível com o tipo de retorno dessa função.

No caso de campos (ou propriedades) ou argumentos do tipo `IEnumerable<R>` estes devem ser iniciados com uma sequência lazy.