07

Aridade de tuplas

Vimos como o compilador emite o código com tuplas por debaixo dos panos. Quando escrevemos o código a seguir:

```
var pessoa = (nome: "Guilherme", empresa: "Alura");
Console.WriteLine(pessoa.nome);
```

O compilador emite:

```
ValueTuple<string, string> pessoa = new ValueTuple<string, string>("Guilherme", "Alura");
Console.WriteLine(pessoa.nome);
```

Aqui, estamos usando o tipo `ValueTuple` com aridade 2, ou seja, a tupla que aceita 2 valores `ValueTuple<T1, T2>`. Contudo, podemos definir tuplas com aridade 3, 4, 5 ou mais:

```
var aridade3 = ("Tenho", 3, "valores");
var aridade4 = ("Eu", "tenho", 4, "valores");
var aridade5 = ("Já", "eu", "tenho", 5, "valores!");
```

Com resultado:

```
ValueTuple<string, int, string> aridade3 = new ValueTuple<string, int, string>("Tenho", 3, "valo

ValueTuple<string, string, int, string> aridade4 = new ValueTuple<string, string, int, string>('

ValueTuple<string, string, string, int, string> aridade5 = new ValueTuple<string, string, string
```

Com estes experimentos, já descobrimos que na biblioteca do .NET existem alguns tipos genéricos de `ValueTuple` já definidos:

```
ValueTuple<T1, T2>
ValueTuple<T1, T2, T3>
ValueTuple<T1, T2, T3, T4>
ValueTuple<T1, T2, T3, T4, T5>
```

Mas será que a quantidade máxima de elementos que podemos ter em uma tupla é limitada ao `ValueTuple` com maior aridade definida dentro da biblioteca do .NET?

Fazendo mais um teste, verificamos que é possível escrever uma tupla com 20 elementos: `(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20)`. Porém, será mesmo que existem todos estes tipos de `ValueTuple` definidos para todas essas 20 aridades?

Na verdade, o `ValueTuple` com maior aridade definido é o de 8 argumentos:

```
ValueTuple<T1, T2, T3, T4, T5, T6, T7, T8>
```

Quando queremos criar uma tupla com 8 ou mais elementos, as coisas ficam interessantes. Uma vez que a aridade máxima é 8, o compilador recorre ao aninhamento de tuples, ou seja, os primeiros 7 elementos são armazenados em suas posições correspondentes na tupla criada e a oitava posição desta tupla armazena o restante (em inglês, rest) em outra tupla!

Por exemplo, para a tuple `(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`, temos o código a seguir (com quebra de linha dos argumentos genéricos para facilitar a leitura):

```
var ultimosElementos = new ValueTuple<
        int,
        int
    >
    (8, 9);

var tuplaCompleta = new ValueTuple<
        int,
        int,
        int,
        int,
        int,
        int,
        int,
        ValueTuple<int, int>
    >
    (1, 2, 3, 4, 5, 6, 7, ultimosElementos);
```

Desta forma, quando executamos `Console.WriteLine(a.Item10);` o código gerado é `Console.WriteLine(valueTuple.Rest.Item3);` – o campo `Rest` é aquele que guarda a outra tupla com os valores restantes.

E essa solução é recursiva. Quando temos uma tupla `experimento` com 15 valores `var experimento = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)`, os primeiros 7 valores ficam em `experimento`, os valores 8 até 14 ficam em `experimento.Rest` e o elemento 15 fica em `experimento.Rest.Rest`.