Module 2

C# 9 records vs F# records

C# evolution

Functional programming languages are often better set up for this: data is immutable (representing information, not state), and is manipulated from the outside, using a freely growable and context-dependent set of functions, rather than a fixed set of built-in virtual methods. Let's continue being inspired by functional languages, and in particular other languages – F#, Scala, Swift – that aim to mix functional and object-oriented concepts as smoothly as possible. Here are some possible C# features that belong under this theme:

- pattern matching
- tuples
- "denotable" anonymous types
- "records" - compact ways of describing shapes
- working with common data structures (List/Dictionary)
- extension members
- slicing
- immutability
- structural typing/shapes?

Mads Torgersen, Program Manager for the C#, January 2015, https://github.com/dotnet/roslyn/issues/98

C# evolution

It's safe to say that the C# design team takes inspiration from other programming languages, including functional ones. In the last several years, C# has received many features previously implemented in F#. For example:

- 2012 - async/await implementation is a port of one, particular implementation available in F# async workflows and turned into a language feature
- 2015 - C# 6 added more features already available in F#, including auto-property initializers, exception filters, expression-bodied function members
- 2017 - C# 7.X - pattern matching, more immutability, value tuples, more expression-bodied members
- 2019 - C# 8 - extended pattern matching, indices and ranges, readonly members
- and C# 9 - pattern matching enhancements, init-only setters and **records**

but also F# has some C# influence, for example, F# Query Expressions were inspired by the LINQ.

and there's nothing wrong with this approach. Looking around and gathering different views allows C# to grow in many directions, making it a true Swiss Army knife.

C# Records

F# Records

If we want to achieve the same result in F#, we can use a record type and declare a variable as mutable.

declaration is used only for demo purposes to achieve non-immutable variable like in C# implementation. Please don't write code like this!

C# Records - with

In the previous lessons, we learned about the expression that can be used to create a clone of the record and apply initializer syntax to modify the properties.

How it may look in F#?

F# Records - with

The F# language also has a mechanism of the record copying expressions and applying a modification syntax on them. Do you see any similarities to the C#?

Except for the copying expression, keyword is also used on other syntax constructions, like pattern matching.

C# Records - equality

As you should already know, the record type provides value-based equality. It means that if two record instances have the same type and their properties are equal, they will be equal.

What about F#?

F# Records - equality

In the F# language, records equality works exactly in the same way. The only difference is the operator since it uses for equality check.

(de)construction syntax

Let's compare syntax for creating positional record and deconstruct operation with F#:

As you may notice, the syntax between these two languages is very similar. In some cases, C# implementation is even shorter and more implicit.

Materials

- https://github.com/dotnet/roslyn/issues/98
- https://blog.ploeh.dk/2015/04/15/c-will-eventually-get-all-f-features-right/
- https://ericbackhage.net/c/new-c-9-features-and-their-f-counterparts/