One flew over the abstraction nest

Dmitri Nesteruk (ret.)
@dnesteruk

Levels of abstraction

1. Machine code
2. Assembly language
3. General-purpose languages
   C, C++, C#, Kotlin
   - C/C++ → ASM, C# → IL, CUDA → PTX
4. Code generation, metaprogramming, DSLs
   ↑ ↑ ↑ what this talk is about

Abstractions are *not* zero-cost

3GL abstractions carry a computational cost

Basic: C++ vs ASM

SIMD wrappers vs. ASM

F# over C#

4th-level abstractions do not necessarily have additional costs (and may have negative costs)

"There Are No Zero-cost Abstractions"
Chandler Carruth, CppCon 2019
https://www.youtube.com/watch?v=rHIkrotSwcc

```
let solveQuadratic a b c =
  let disc = b * b - 4.0 * a * c
  let calc op = (op (-b) (sqrt disc)) / (2.0*a)
  (calc (+), calc(-))
```

Enums: a case study

Enums in C# are of ‘system programming’ variety

Enums in (Java, Rust, …) are of the ‘concrete instances of class’ variety

Can we uplift C# enums?

```
public enum Color
{
  Red, Green, Blue
}

public void SetColor(Color color)
{
  ...
}
```

Enums are great!

Strongly typed, integer based (can force-cast)

`Enum.GetValues()` gets you all the cases

Adding a new enum member does not break code

Very finite amount of data storage (1 member)

```
public enum Color
{
    Red   = 0xff0000,
    Green = 0x00ff00,
    Blue  = 0x0000ff,
}
```

```
public enum Color
{
    Red   = 0xff0000,
    Green = 0x00ff00,
    Blue  = 0x00ff00,
    DarkBrown = 0x654321
}
```

```
public static class ColorExtensions
{
  public static string GetName(this Color c)
  {
    switch (c)
    {
      case Color.Red: return "red";
      case Color.Green: return "green";
      case Color.DarkBrown: return "dark brown";
    }
    return "unknown;"
  }
}
```

Problems

Want to store additional data

Don’t want to work with compactified data

Want member access, i.e., color.R

Want code to be kept together (i.e., member methods instead of extension methods)

Basically, need an ordinary class with a couple of static, readonly members

Demo

Enums.cs

Color formats

```
type Color =
    Red
    | Green
    | Blue
    | RGB of r: byte * g: byte * b: byte
    | CMYK of c: byte * m: byte * y: byte * k: byte
```

Demo

Fonts.cs

Leveraging SIMD

.NET Core supports SIMD (SSE) instructions

Allow multiple arithmetic operations in parallel

Needs data locality

SoA/AoS problem

Roslyn to the rescue!

```
public struct GameObject
{
    public Point Position;
    public Vector Location;
}
GameObject [] objects =
    new Pixel[100];
```

```
public struct GameObjects
{
  public Point Positions[];
  public Vector Locations[];
  public Pixels(int size)
  {
    Positions = new Point[size]; // etc.
  }
}
GameObjects objs = new GameObjects(100);
objs[0].Positions.X = 0; // how?
```

Roslyn to the rescue!

Parse existing class

Identify all fields

Create corresponding arrays + plumbing (constructors etc.)

Create indexer + associated proxy

Demo

Blackmire

Conclusion

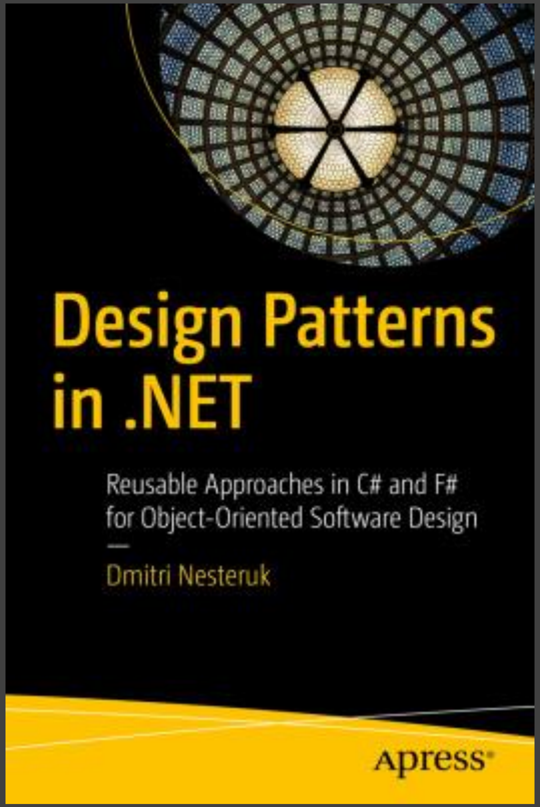
When raising the level of abstraction, you balance usability and performance costs

Code generation mechanisms (T4, Roslyn-based, etc.) can be truly zero-cost

That’s it!

Questions? Answers? Hate mail? @dnesteruk

Design Patterns in .NET online course at http://bit.ly/2p3aZww


Design Patterns in .NET

Reusable Approaches in C# and F# for Object-Oriented Software Design

Dmitri Nesteruk

Apress