Ayman Anaam • 2nd
Dynamic Technology Leader | Innovator in .NET Development and Cloud Solutions
2h • Edited •

Follow

Async/Await in C#: Patterns, Pitfalls, and Performance

Writing scalable, responsive C# apps isn't just about using async/await, it's about using it correctly.

Here's a quick guide:

✅ Key Benefits:

- Non-blocking execution (better UI & server scalability)
- Simplified code (goodbye, callback hell!)

🧠 Common Patterns:

- await DoWorkAsync() for non-blocking flows
- Task.WhenAll() for parallel execution

⚠️ Critical Anti-Patterns:

❌ Blocking async code with .Result or .Wait() → causes deadlocks

❌ Using async void → crashes your process on unhandled exceptions

❌ Ignoring returned Task objects → silent failures

🚀 Performance Tips:

- Use ConfigureAwait(false) in libraries
- Prefer ValueTask for hot paths
- Avoid excessive parallelism to prevent thread starvation

🔍 Advanced Patterns:

- Proper CancellationToken usage
- Timeout patterns with CancellationTokenSource

🛠️ Best Practices Checklist:

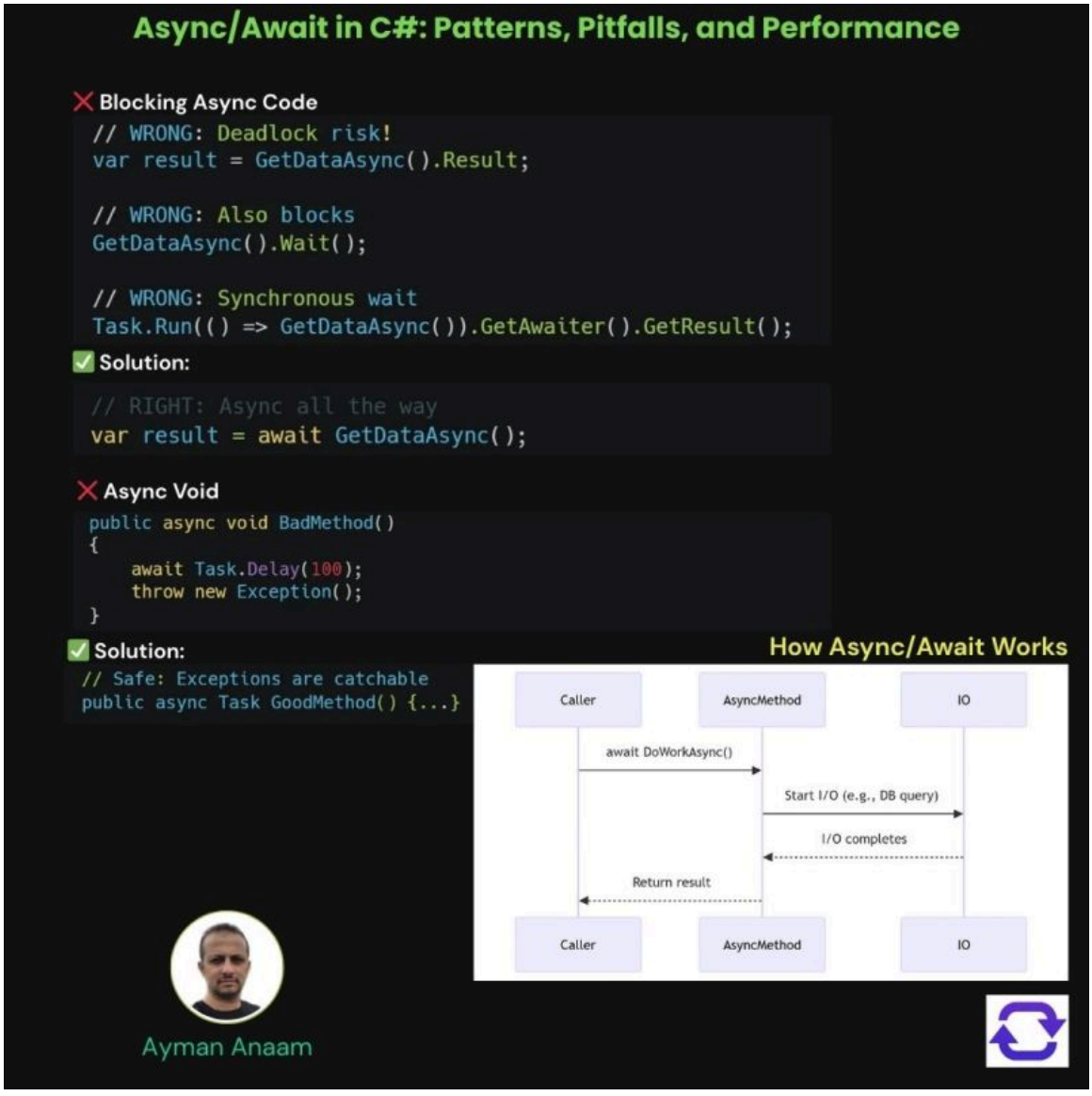
- Always await tasks.

- Never block async code.
- Catch exceptions properly.

Want a complete guide with code samples and diagrams?

Check out the full article here: https://shorturl.at/C2VK7

#dotnet #csharp #asyncawait #programming #softwareengineering



14 1 comment 4 reposts

Like Comment Repost Send