What Happens if You Don't Dispose?



Elton StonemanConsultant & Trainer

@EltonStoneman blog.sixeyed.com



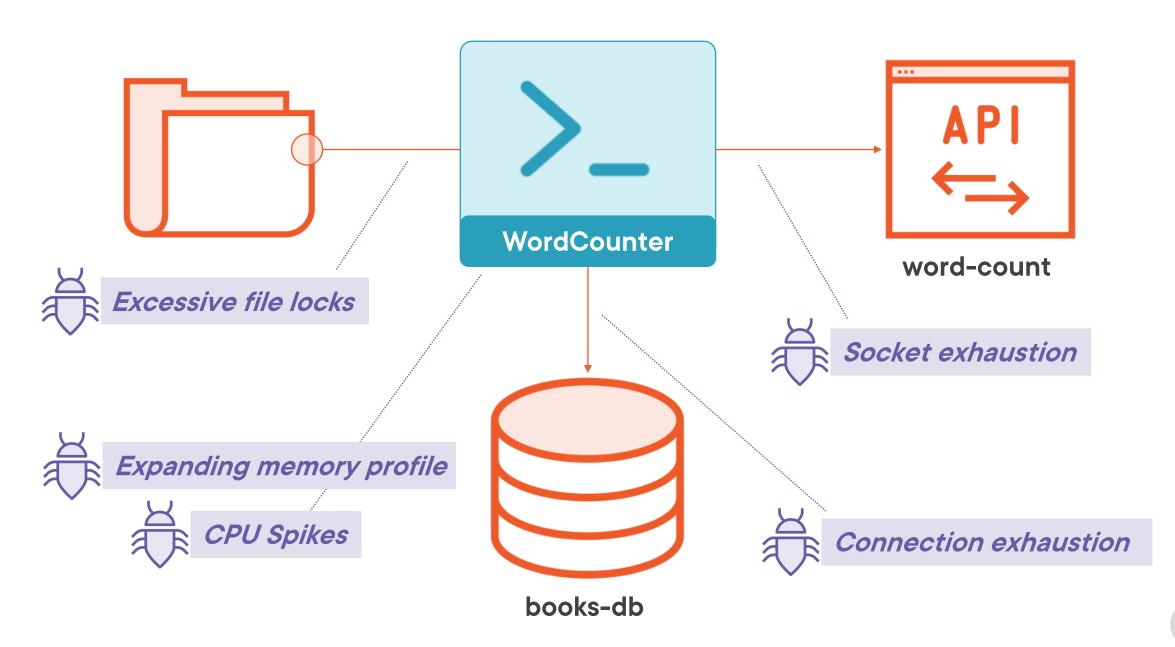
Module Outline

Finding disposable issues

Disposing in modern .NET apps

Fixing disposable problems







Coding

Static Analysis

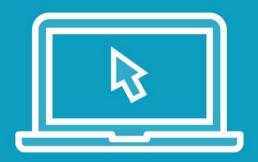
Profiling

Testing





Demo



Finding problems with object lifetimes

- Debugging suspicious areas
- Watching file IO
- Load testing with the profiler



Demo

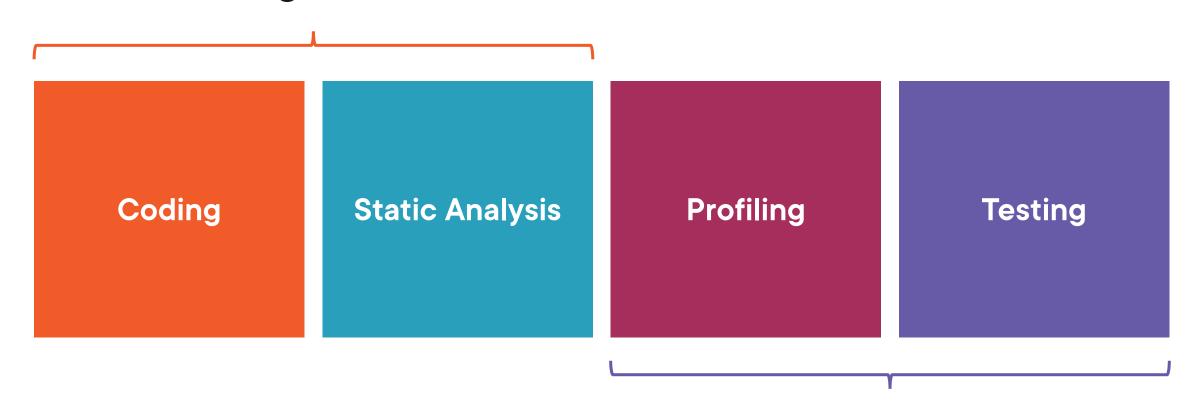


Fixing problems with object lifetimes

- Static analysis in Visual Studio
- IDisposable usage rules
- Load testing with the profiler



Design-time



Run-time



Static Code Analysis

Enabled in the project file properties

CA2000

Dispose objects before losing scope

* https://is.gd/usuzas



Fixing CA2000

FileArchiver.cs

```
// before
var inputStream = File.OpenRead(sourcePath);
var outputStream = File.Create(targetPath);
inputStream.CopyTo(outputStream);
// after
using var inputStream = File.OpenRead(sourcePath);
using (var outputStream = File.Create(targetPath))
  inputStream.CopyTo(outputStream);
```

SqlClient.cs

```
Missing CA2000
```

```
// flagged
var sqlConnection = await OpenConnection();
// not flagged
var cmd = sqlConnection.CreateCommand();
// so you need to know the domain
using var sqlConnection = await OpenConnection();
using (var cmd = sqlConnection.CreateCommand())
```

ApiClient.cs

```
public class ApiClient : IDisposable
 private HttpClient _httpClient = new();
 protected virtual void Dispose(bool ...)
    if (disposing && _httpClient != null)
       _httpClient.Dispose();
       httpClient = null;
```

Program.cs

```
var apiClient = new ApiClient(_Config);
var lines = File.ReadAllLines(path);
for (var i = 0; i < lines.Length; i++)</pre>
  // create tasks using apiClient
finally
  apiClient.Dispose();
```

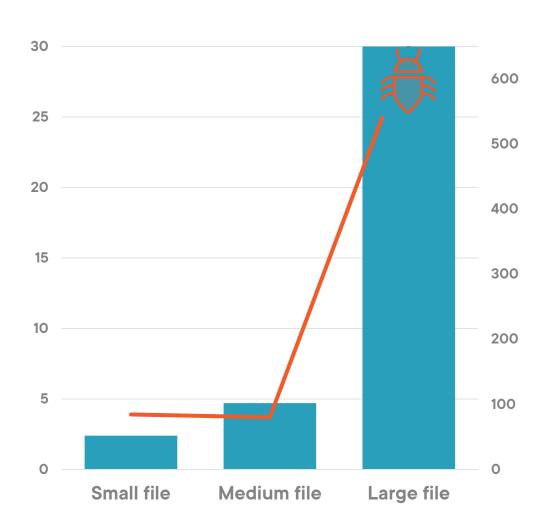
Disposing Tasks

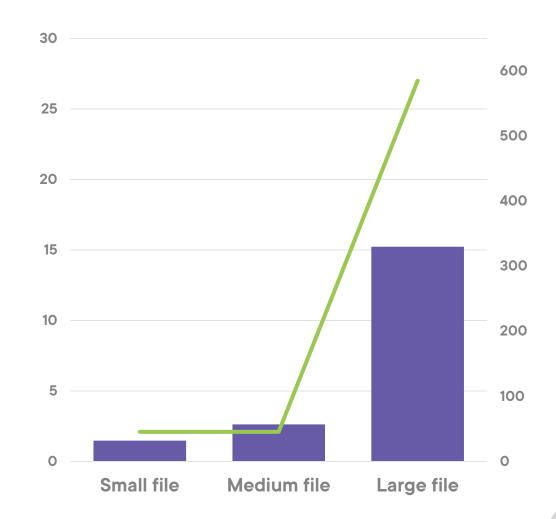
Program.cs

```
for (var i = 0; i < lines.Length; i++)</pre>
  apiTasks.Add(Task.Run(async () => await //...
try
  Task.WaitAll(apiTasks.ToArray(), cts.Token);
finally
  apiTasks.ForEach(x => x.Dispose());
```

Before

After



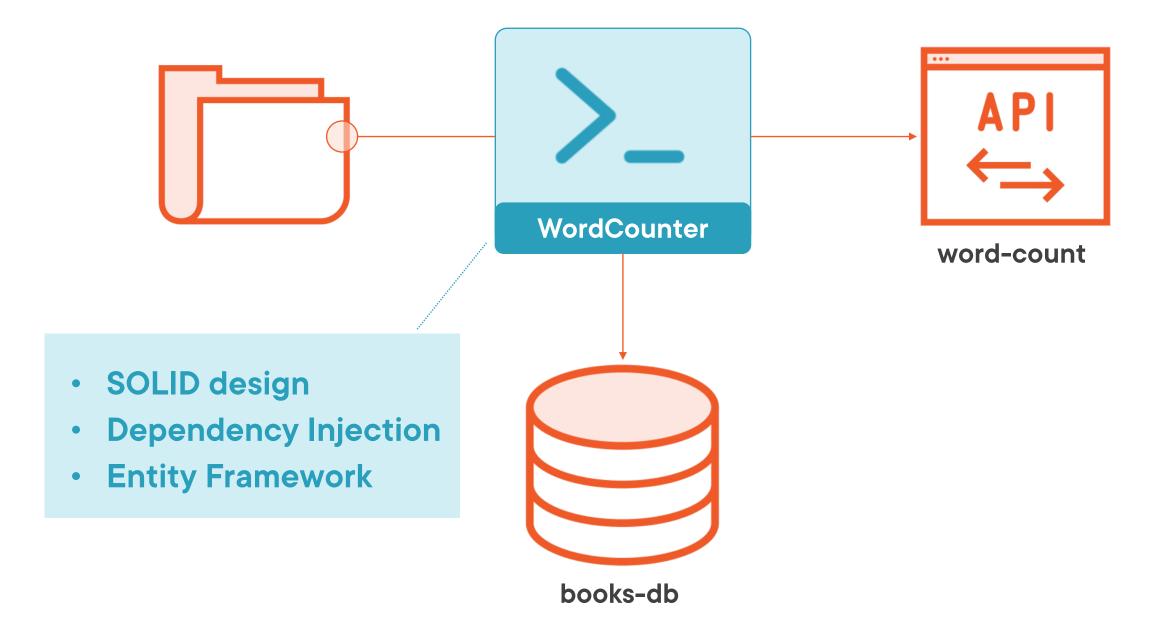




Best Practice #6

Enable static analysis with rule CA2000







Demo



Managing object lifetime in modern apps

- Dependency injection
- Code walkthrough
- Load testing with the profiler



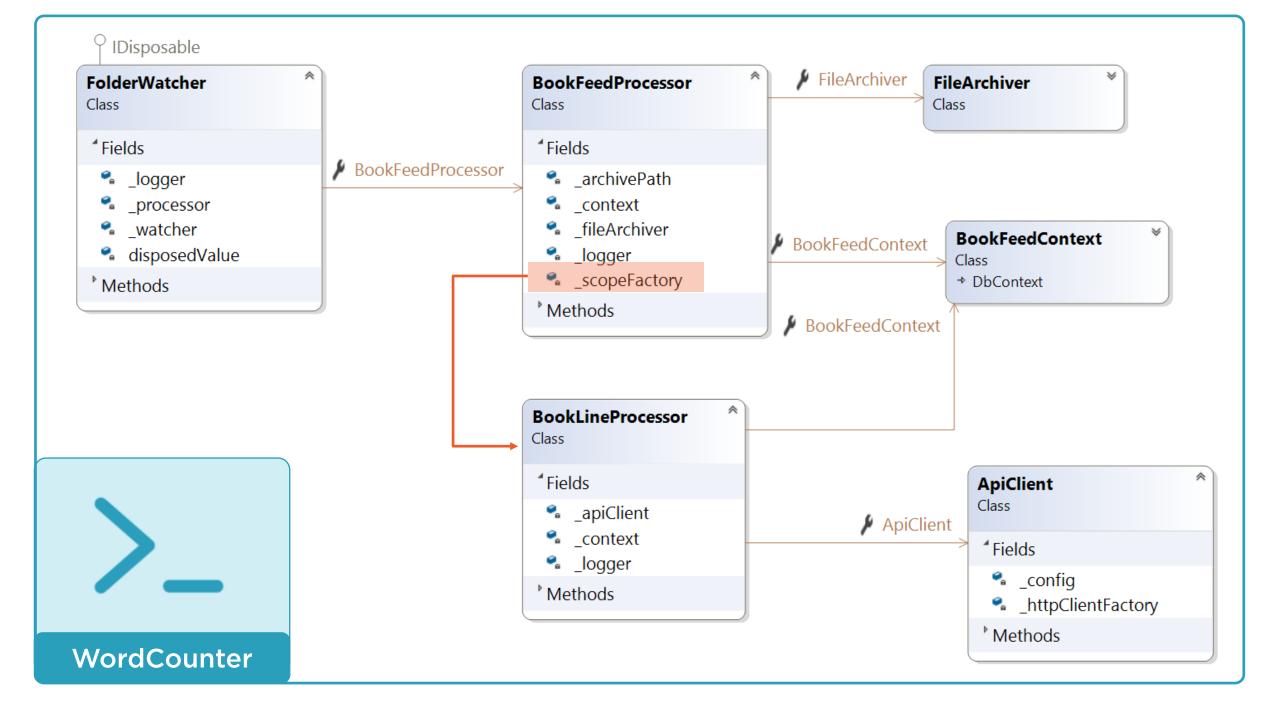
Demo



Fixing object lifetime in modern apps

- CA2000 analysis
- Scope lifetimes
- Disposable special cases





```
apiTasks.Add(Task.Run(async () =>
{
   using (var scope = _scopeFactory.CreateScope())
   {
     var processor = scope.ServiceProvider.GetRequiredService<BookLineProcessor>();
     return await processor.GetWordCount(path, lineNumber, line, cancellationTokenSource);
   }
}));
```

Dependency Scopes

Explicit scope creation for tasks

```
private readonly BookFeedContext _context;
private readonly ApiClient _apiClient;

public BookLineProcessor(BookFeedContext context, ApiClient apiClient)
{
    _context = context;
    _apiClient = apiClient;
}
```

DbContext

Instance per scope or transient - not disposed

```
private IHttpClientFactory _httpClientFactory;

public ApiClient(IHttpClientFactory httpClientFactory)
{
    _httpClientFactory = httpClientFactory;
}
// ...
var httpClient = _httpClientFactory.CreateClient();
```

HttpClient

Lifetime managed by HttpClientFactory

Service Registration

Program.cs

```
// config, logging & HttpClientFactory
var services = new ServiceCollection()
       .AddSingleton(_Config)
       .AddLogging( //... )
       .AddHttpClient();
// DbContext
services.AddDbContext<BookFeedContext>(
 options => options.UseSqlServer //... ),
 ServiceLifetime.Transient);
```

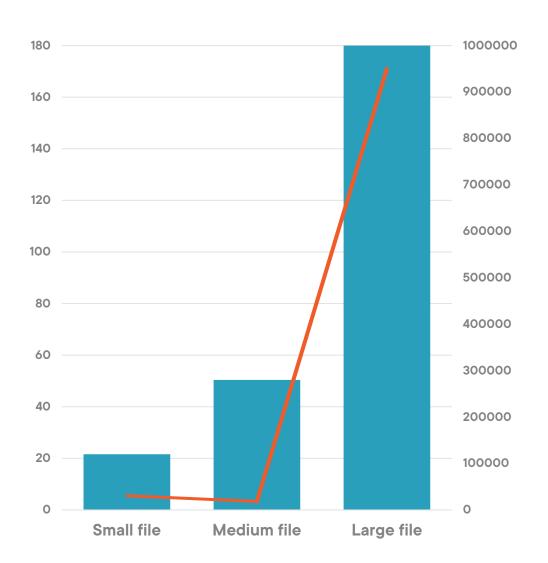
Best Practice #7

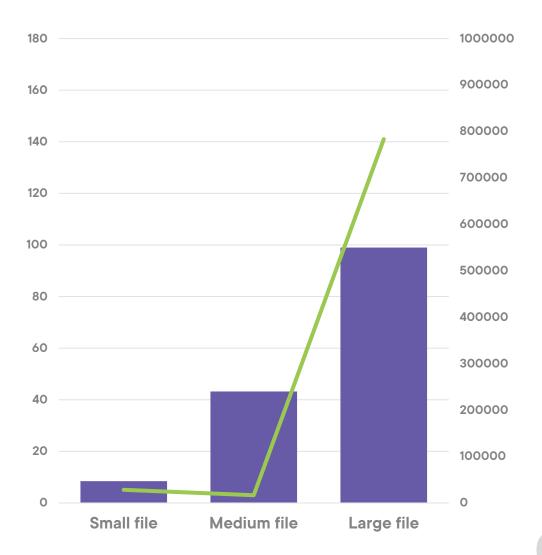
Know your domain:)



Before

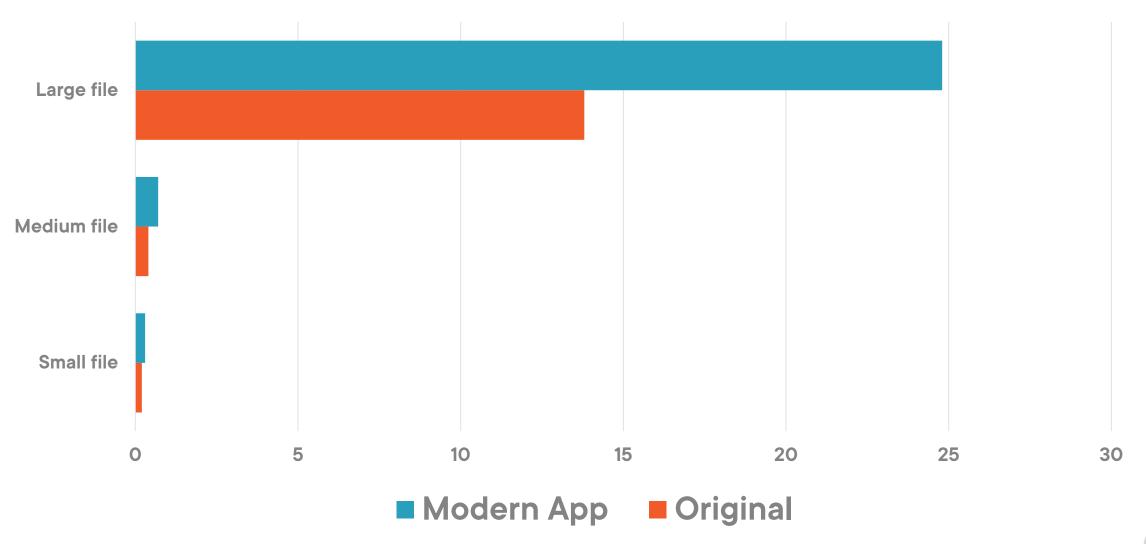
After







The Cost of Modernization





IDisposable

```
namespace System
   // Provides a mechanism for releasing unmanaged resources.
    public interface IDisposable
        // Performs application-defined tasks associated with
        // freeing, releasing, or resetting unmanaged resources.
        void Dispose();
```



lAsyncDisposable

```
namespace System
 // Provides a mechanism for releasing unmanaged resources asynchronously.
  public interface IAsyncDisposable
    // Performs application-defined tasks associated with
   // freeing, releasing, or resetting unmanaged resources asynchronously.
    ValueTask DisposeAsync();
```



Demo



Disposable Async Resources

- Implementing IAsyncDisposable
- Working with asynchronous streams
- Using async disposables



```
await using (var rand = new RandomStringGenerator())
{
    await foreach (var s in rand.Get(50))
    {
        Console.WriteLine(s);
    }
}
```

Consuming IAsyncDisposables

Streaming sources - gRPC

lAsyncDisposable

RandomStringGenerator.cs

```
public class RandomStringGenerator : IAsyncDisposable
   private MemoryStream buffer = new(100 * 1024 * 1024);
   public async IAsyncEnumerable<string> Get() { // ... }
    public async ValueTask DisposeAsync()
       await DisposeAsyncCore();
       GC.SuppressFinalize(this);
    protected virtual async ValueTask DisposeAsyncCore()
       if (_buffer is not null)
            await buffer.DisposeAsync().ConfigureAwait(false);
           buffer = null;
```

Best Practice #8

Implement IAsyncDisposable if your class uses an async disposable field



Summary



Finding and fixing disposable issues

- Static analysis
- Memory profiling
- Domain knowledge

Object lifetime in modern apps

- Dependency injection scopes
- Understanding key classes

Asynchronous streams

- Implementing IAsyncDisposable
- Using async disposables



Up Next:

Just the Best Practices

