IDisposable Best Practices for C# Developers

Introducing IDisposable



Elton StonemanConsultant & Trainer

@EltonStoneman blog.sixeyed.com

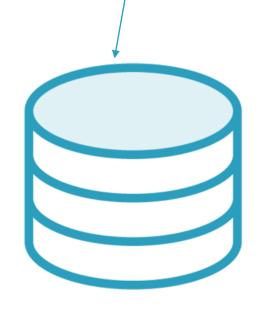


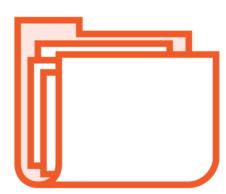
.NET Framework 1.0 - 4.8

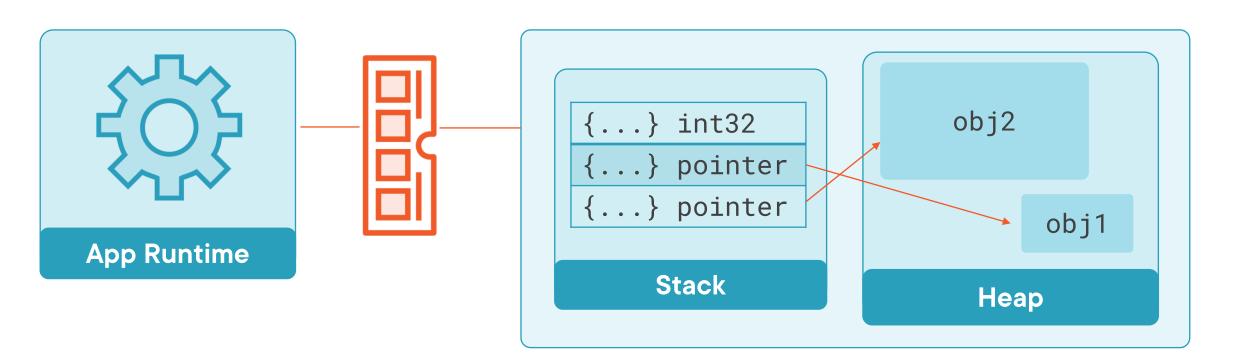
.NET Core 1.0 - 3.1

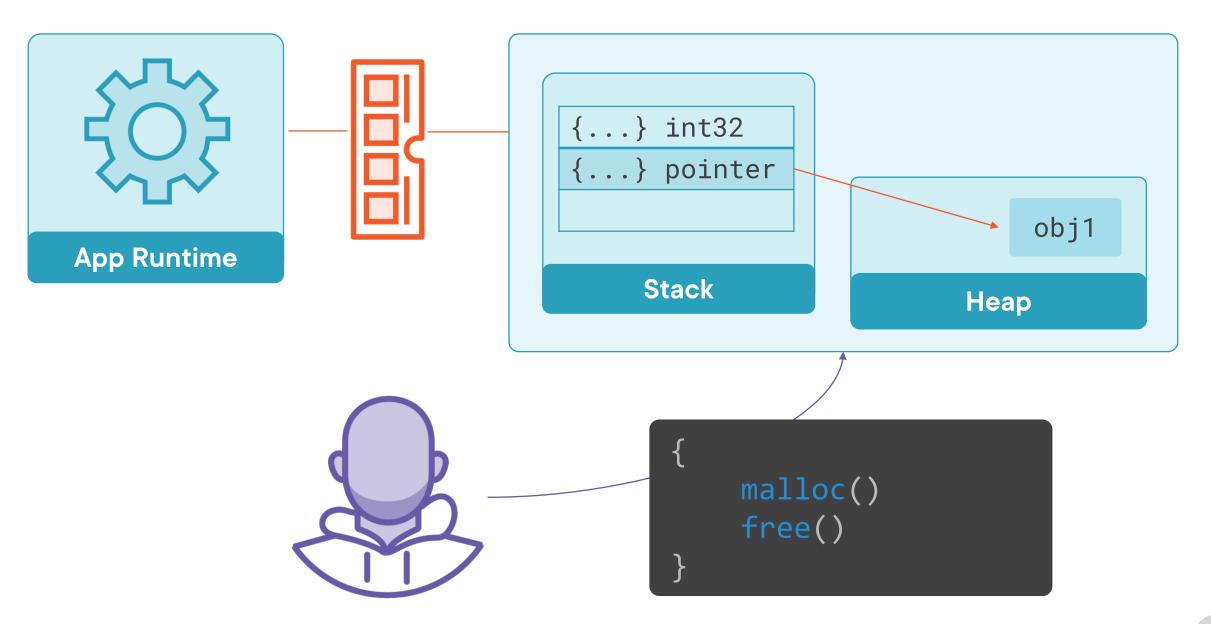
.NET 5+

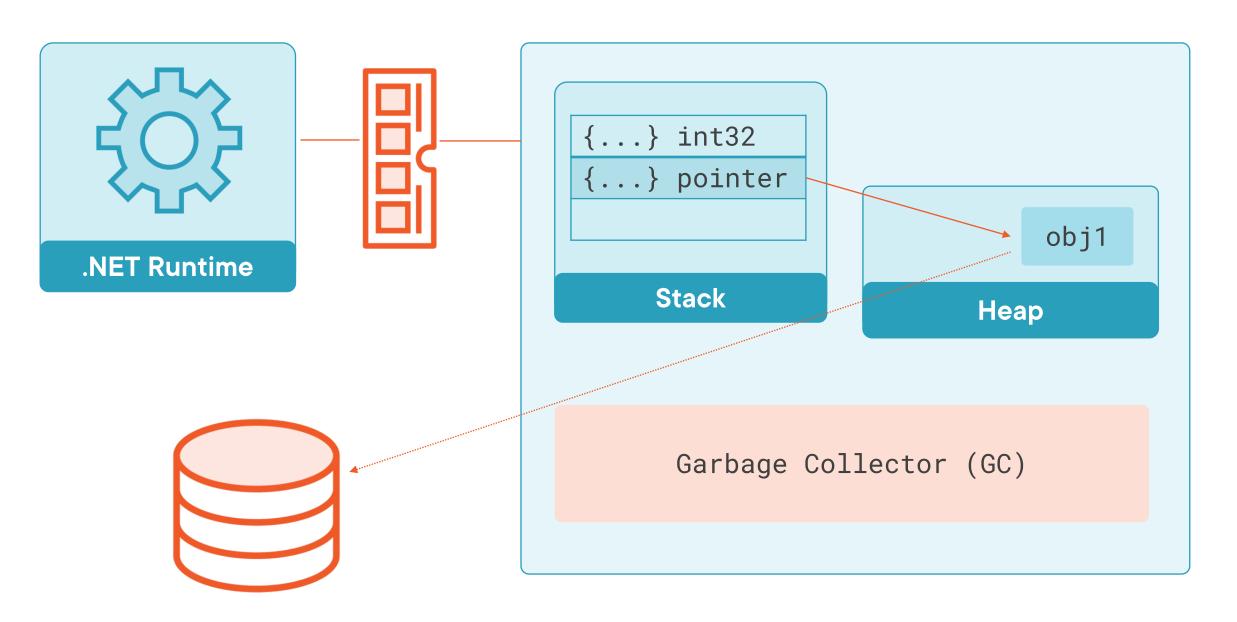












Not Using IDisposable?







Locking external resources



Functional defects



<<interface>>

IDisposable

+ Dispose()

Introducing IDisposable

- Understanding the interface
- Working with disposable objects
- Implementing IDisposable
- All .NET runtimes



Course Outline

Introducing IDisposable

What happens when the Garbage Collector runs?

What happens if you don't dispose?



Marker Interfaces

```
IDoNothing.cs
```

```
interface IDoNothing
{
}
```

DoesNothing.cs

```
class DoesNothing : IDoNothing
{
}
```

Simple Interfaces

IDoOneThing.cs

```
interface IDoOneThing
{
    void DoTheThing();
}
```

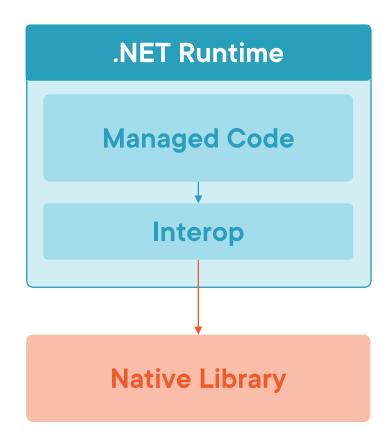
DoesNothing.cs

```
class DoesOneThing : IDoOneThing
{
    void DoTheThing()
    {
    }
}
```

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();
```



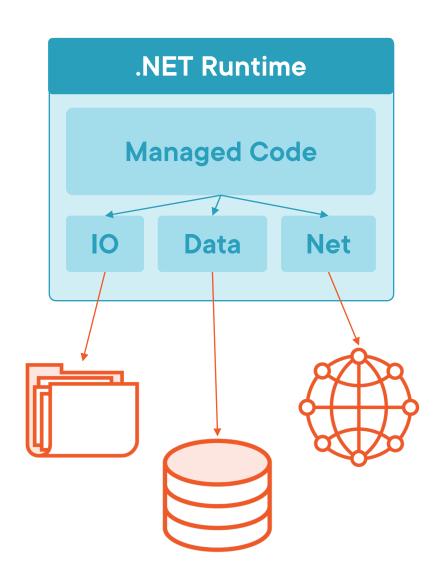


Runtime-Callable Wrapper (RCW)

- Interface with COM objects
- .NET on Windows

Explicitly unmanaged

- Dilimport
- IntPtr

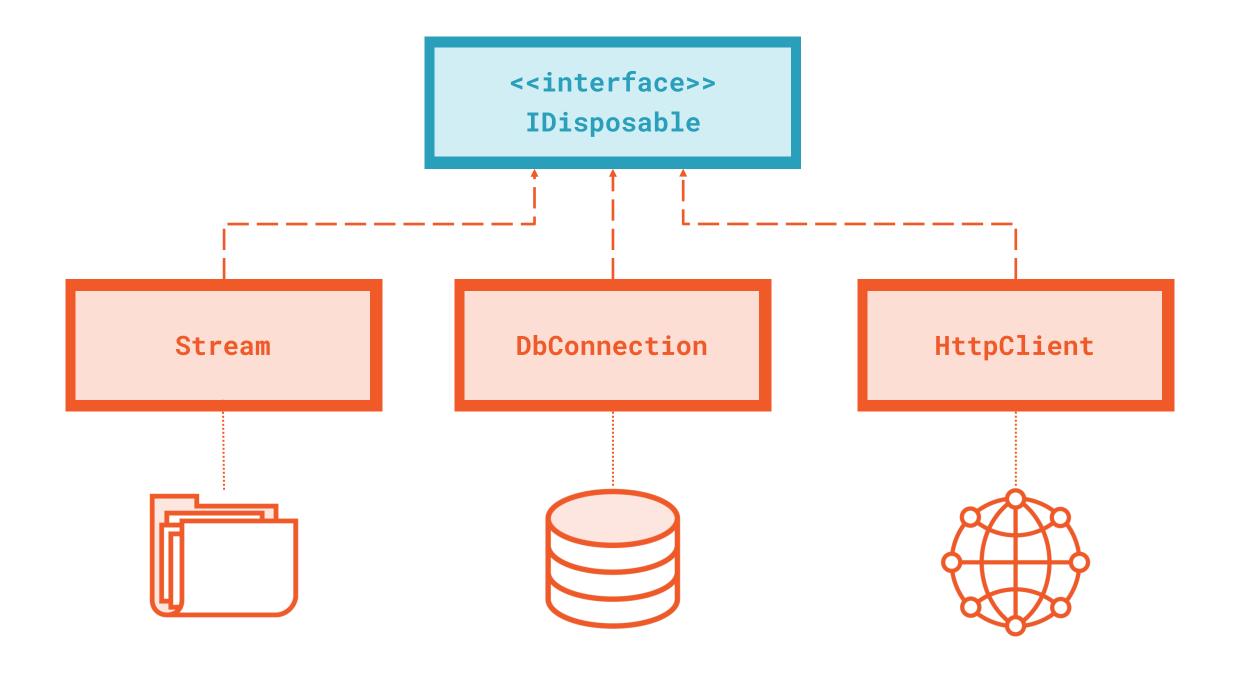


Managed access

- External resources

Implicitly unmanaged

- System.IO
- System.Data
- System.Net



Using IDisposable

Custom.cs

```
public class Custom : IDisposable
{
    public void Method() {}

    public void Dispose() {}
}
```

Program.cs

```
static void Main()
{
    using (var obj = new Custom())
    {
       obj.Method();
    }
}
```

Using IDisposable

Custom.cs

```
public class Custom : IDisposable
{
    public void Method() {}

    public void Dispose() {}
}
```

Program.cs

```
static void Main()
{
    using (var obj = new Custom())
    {
       obj.Method();
    }
}
```

```
static void Main()
{
    using (var obj = new Custom())
        obj.Method();
   // or:
    using var obj = new Custom();
    obj.Method();
```

◄ Disposable object declaration

■ End of scope - object disposed

- Alternative without braces
- Scope not always clear

```
static void Main()
   // using var obj = new Custom();
   // obj.Method();
   var obj = new Custom();
    try
        obj.Method();
   finally
        obj.Dispose();
```

◄ Functionally equivalent

■ No magic :)

```
static void Main()
   // using var obj = new Custom();
   // obj.Method();
   var obj = new Custom();
   try
        obj.Method();
   finally
        // obj.Dispose();
```

■ Not good

Demo



Working with disposable objects

- SqlConnection for data access
- Disposing and not disposing
- Monitoring resources



Demo



Working with disposable objects

- Dispose failures under load
- Resource exhaustion
- Application exceptions



Using IDisposable

DatabaseState.cs

```
class DatabaseState : IDisposable
{
    private SqlConnection _cx;

    /* ... */
}
```

Test.cs

```
using (var s = new DatabaseState())
{
    s.GetDate;
}
```

Implementing IDisposable

DatabaseState.cs

```
public class DatabaseState : IDisposable
    private SqlConnection _connection;
   /* · · · */
    public void Dispose()
        _connection.Close();
        _connection.Dispose();
```

```
using (var command = _connection.CreateCommand())
{
    command.CommandText = "SELECT getdate()";
    return command.ExecuteScalar().ToString();
}
```

Managing scope

Dispose called when method returns

Best Practice #1

Dispose of IDisposable objects as soon as you can



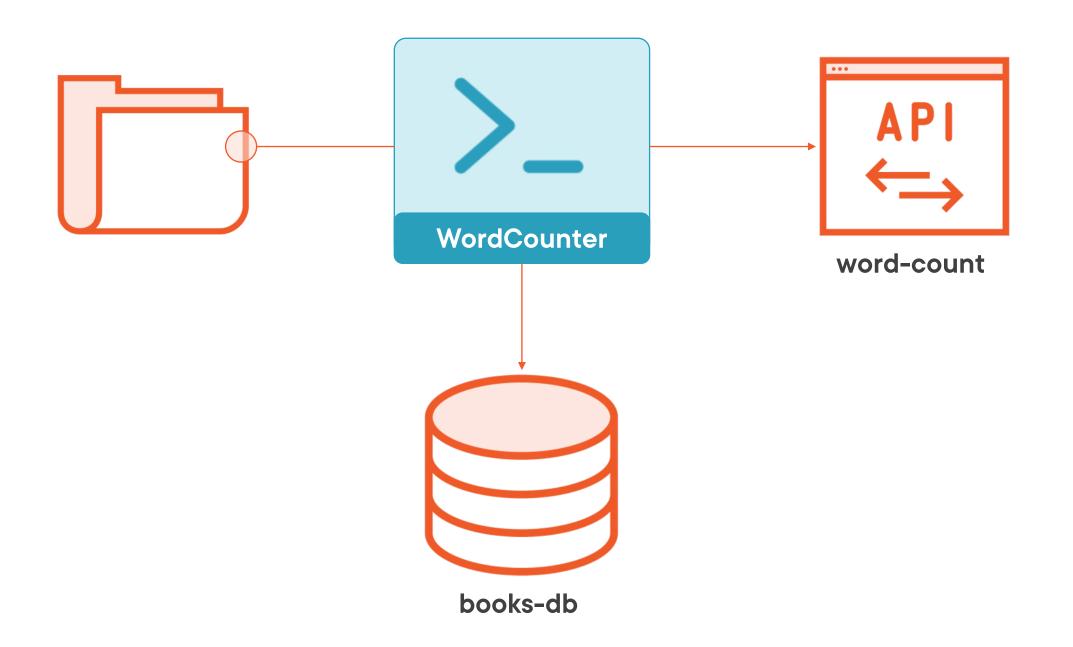
Disposing Disposable Objects

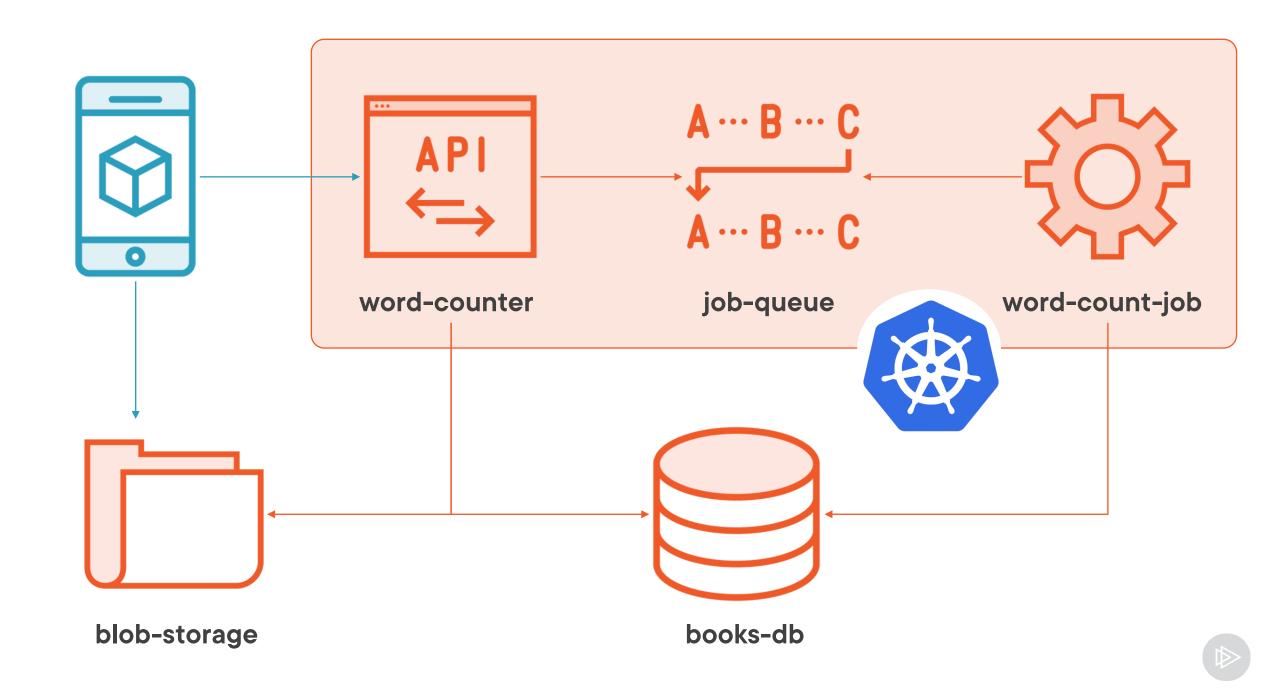
Typical.cs

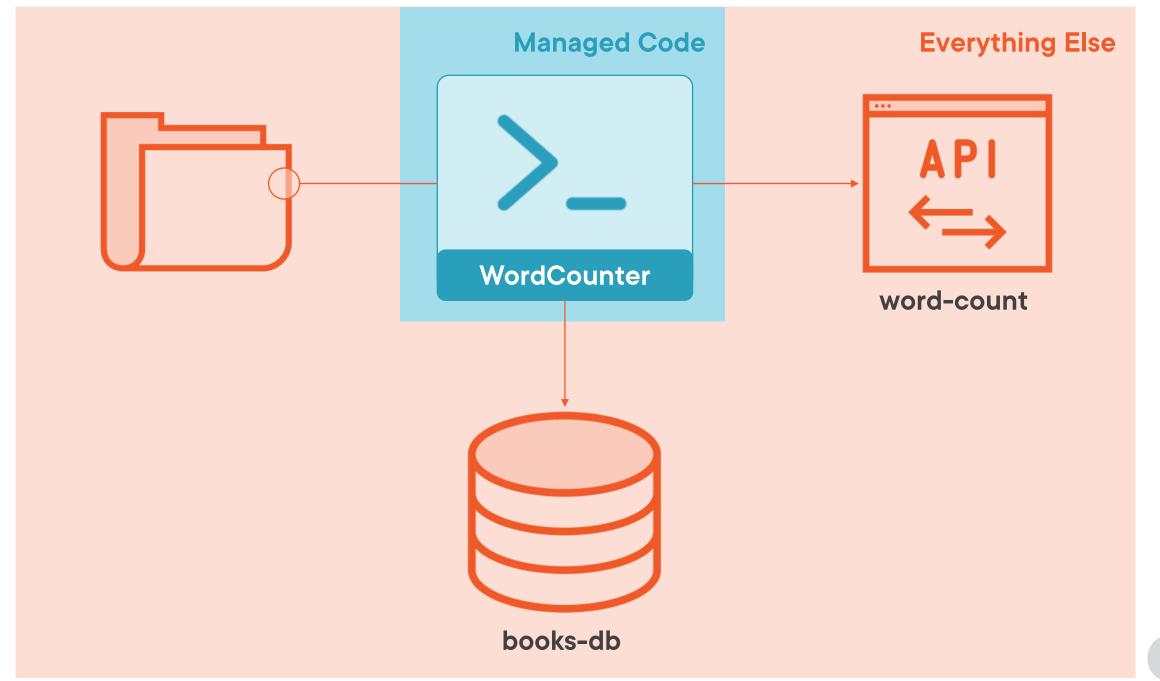
```
using (var obj = new Custom())
{
    // work with obj
}
// obj.Dispose() is called here
```

Alternative.cs

```
finally
    try
        obj.Dispose();
    catch (Exception ex)
        // handle ex
```









Demo



Consuming resources without disposing

- The WordCounter app
- Proving basic functionality
- Understanding the architecture



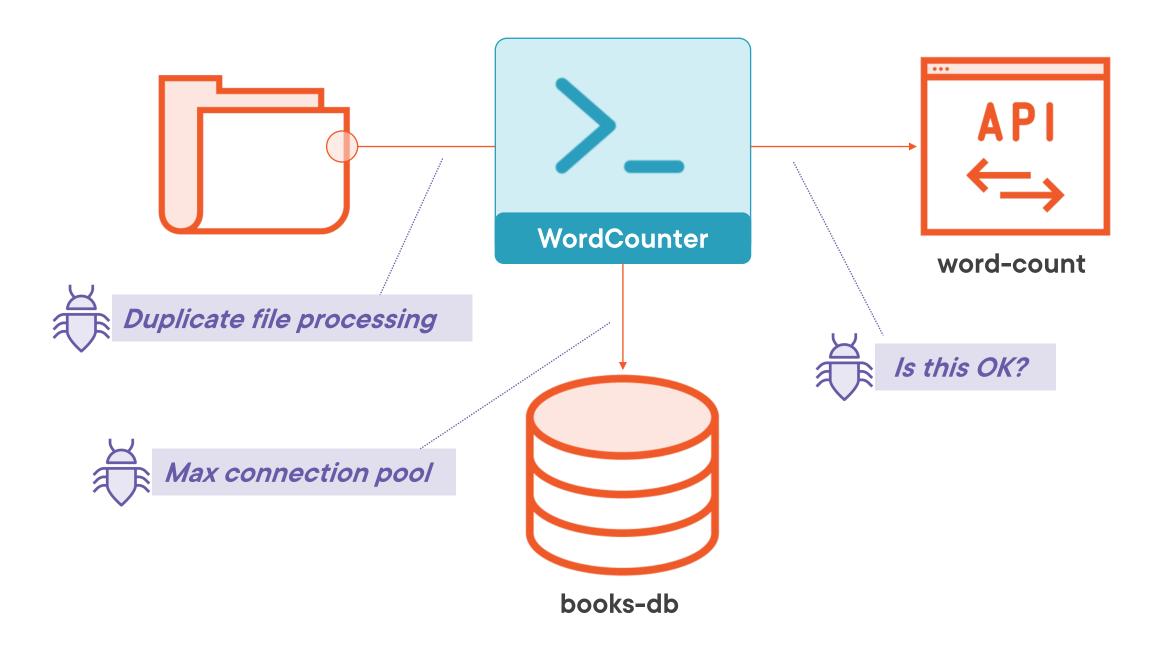
Demo



Consuming resources without disposing

- Finding problems
- Resource exhaustion
- Functional defects







Summary



IDisposable

- Explicit lifetime managment
- Freeing up resources

Unmanaged resources

- Used explicitly
- Used in managed resources
- Files, databases, HTTP services

Problems with not disposing

- Resource exhaustion
- Memory consumption
- Environment-specific defects



Up Next:

What Happens When the Garbage Collector Runs?

