What Happens When the Garbage Collector Runs?



Elton StonemanConsultant & Trainer

@EltonStoneman blog.sixeyed.com

Module Outline

What is the Garbage Collector?

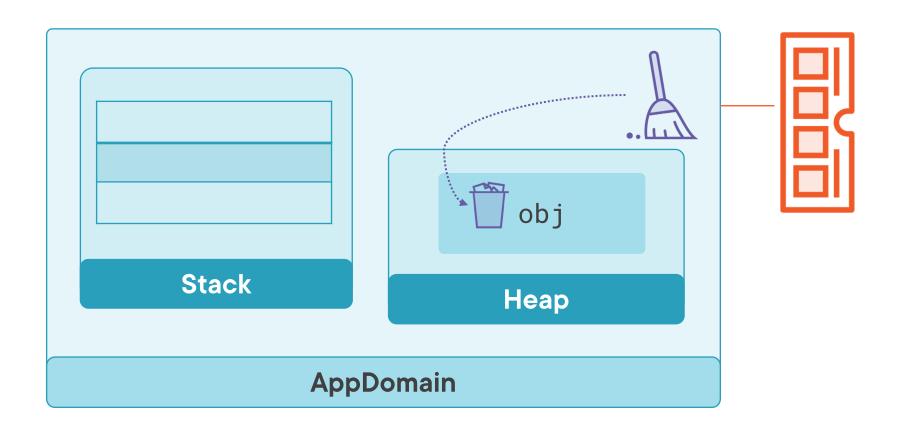
How the Garbage Collector cleans up objects The best practice implementation of IDisposable



NET's garbage collector manages the allocation and release of memory for your application.

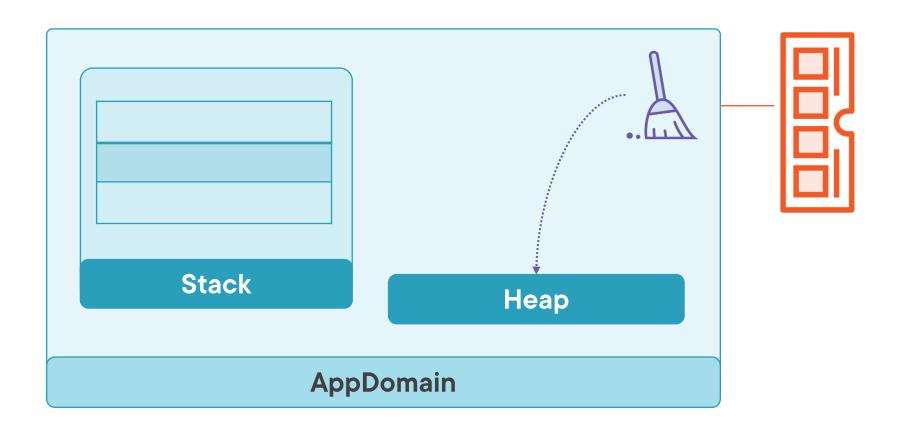


var obj = new Custom();





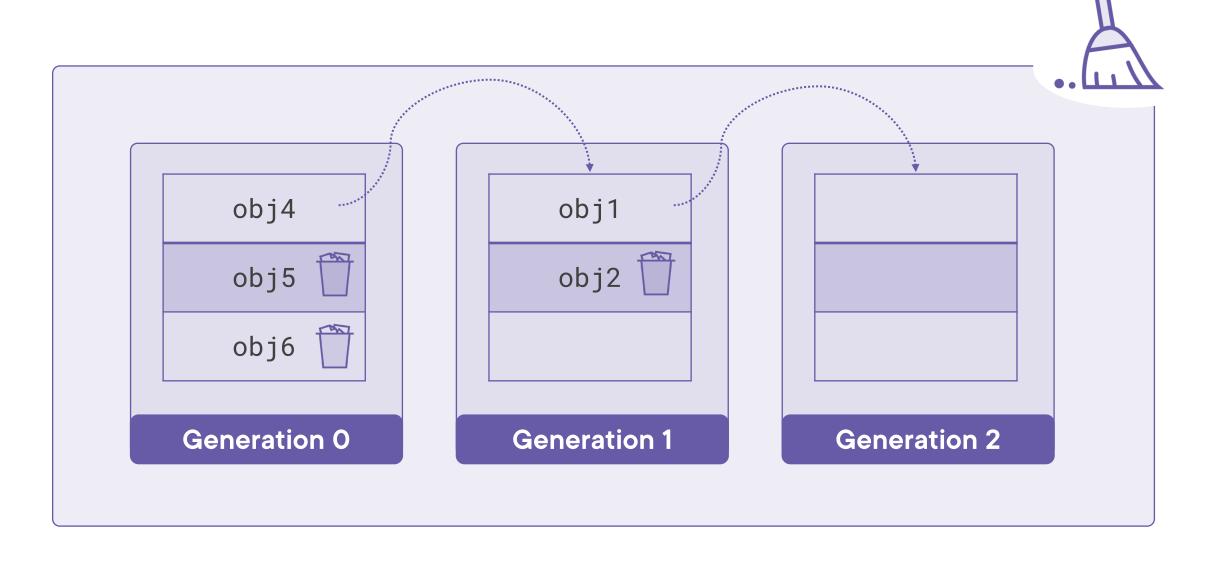
var obj = new Custom();





Garbage Collector Generations







obj7 obj8

obj9

Generation 0

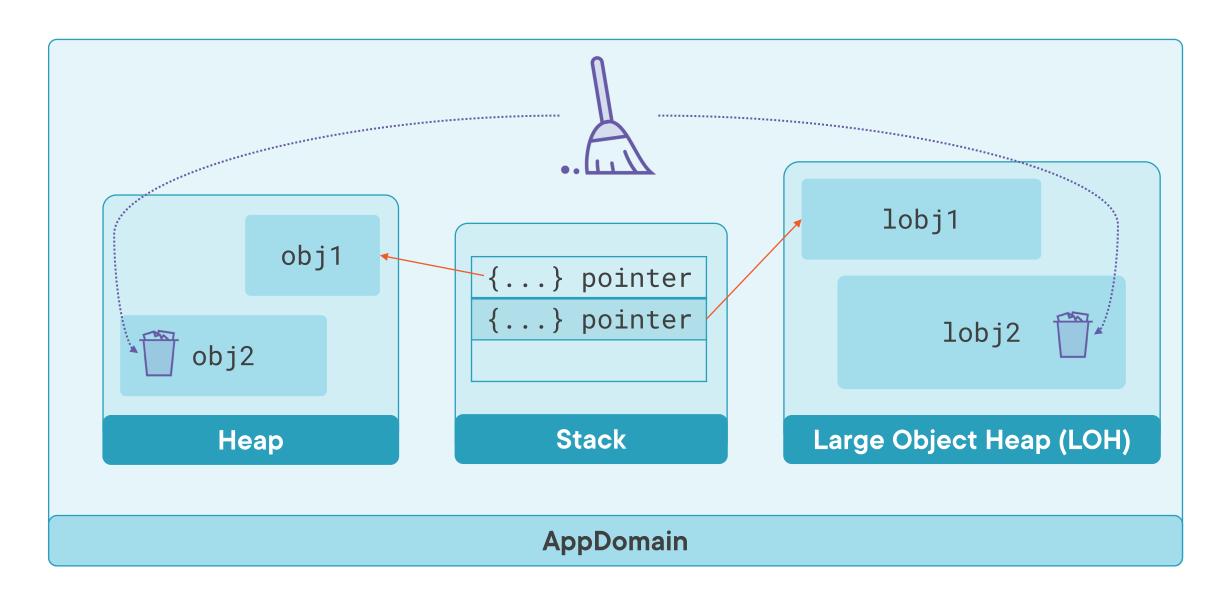
obj4

Generation 1

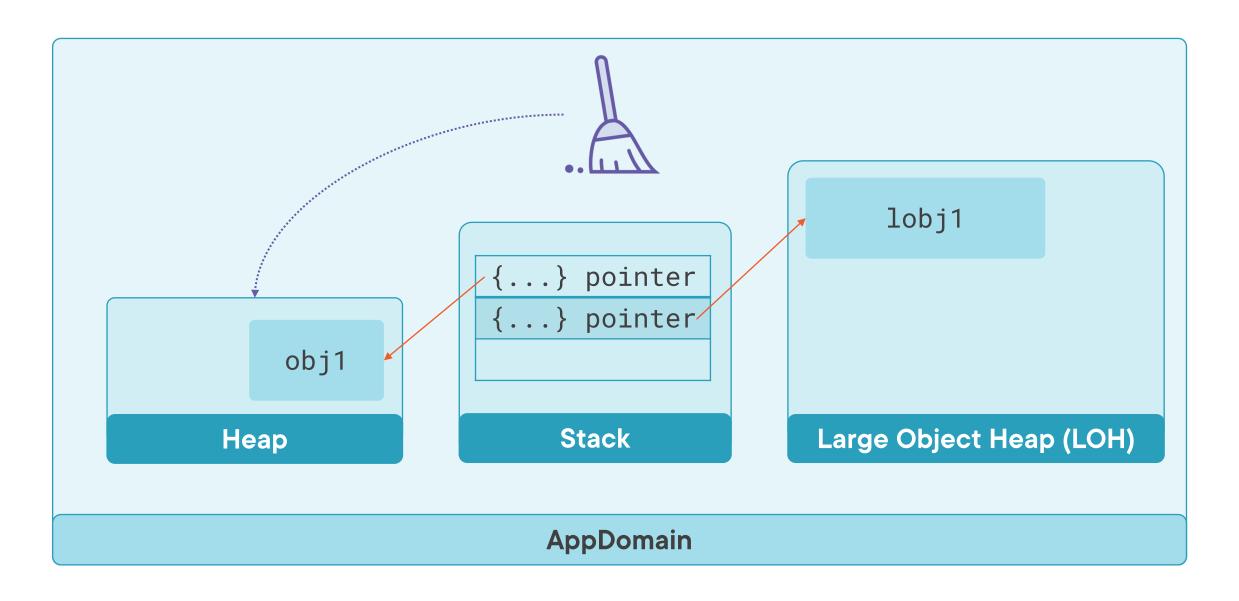
obj1

Generation 2

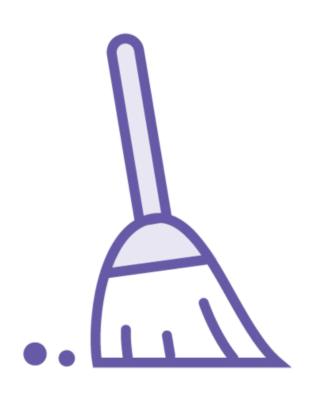












Memory

- Available
- Contiguous

Collections

- Efficient
- Infrequent



When Does the GC Collect?







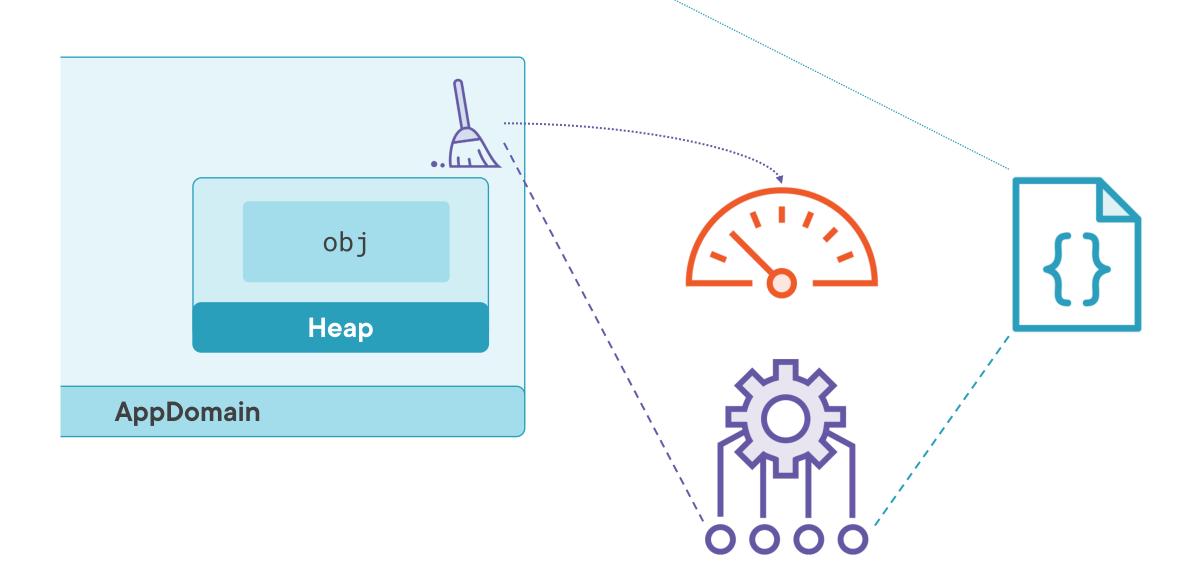
Generation 0

Generation 0 + Generation 1

Generation 0 + Generation 1 + Generation 2



var obj = new Custom();





Demo



Understanding Garbage Collection

- Undisposed static fields
- Profiling .NET objects
- Forcing the GC to run

Demo



Understanding Garbage Collection

- Undisposed local variables
- Not disposing SqlConnection
- Checking for object leaks

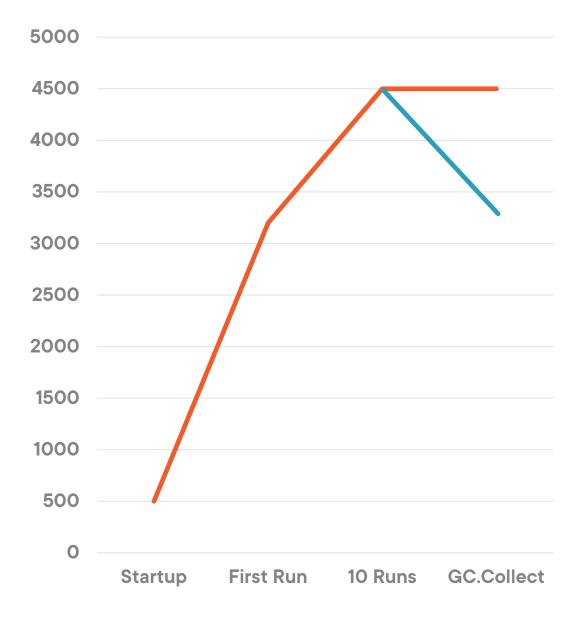


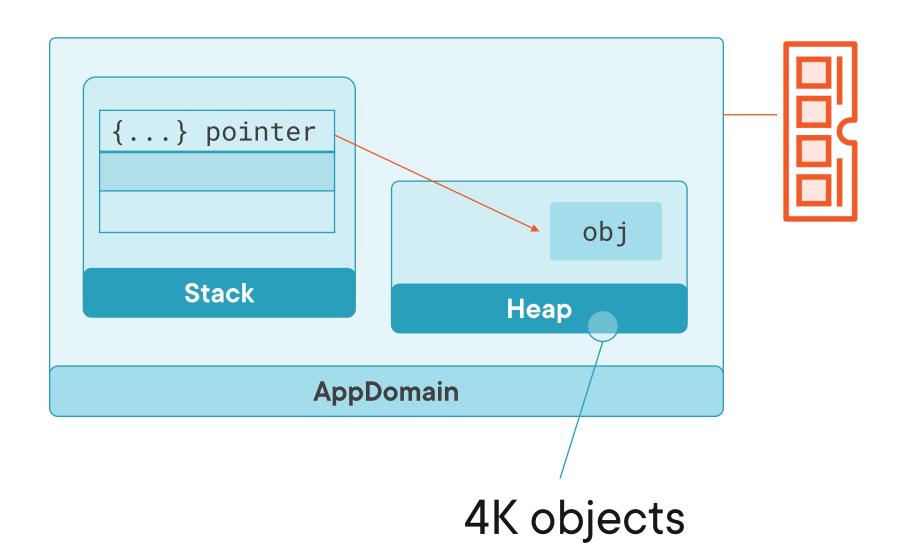
```
# initialize static field
if ( DatabaseState == null)
  DatabaseState =
    new DatabaseState( Config);
# print output
Console.WriteLine(
  DatabaseState.GetDate());
```



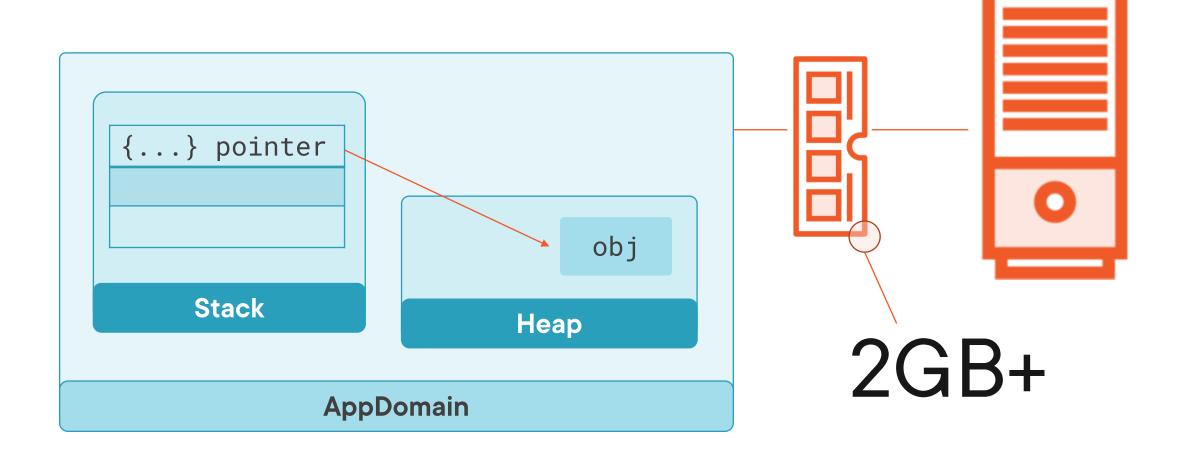
```
# create a new object each time
var s =
  new DatabaseState(_Config));
Console.WriteLine(s.GetDate());
```

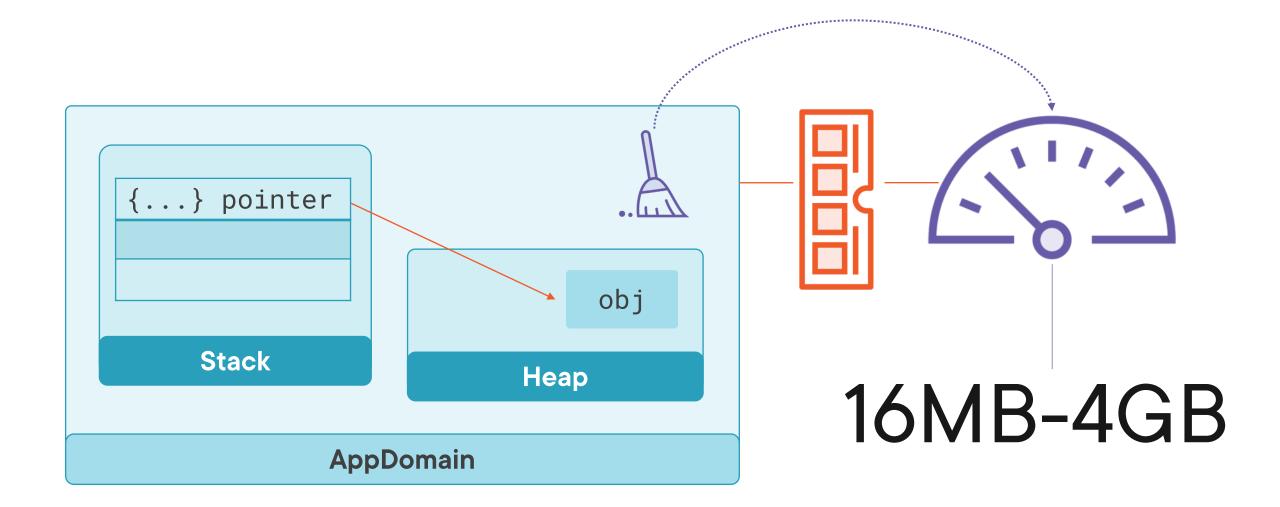


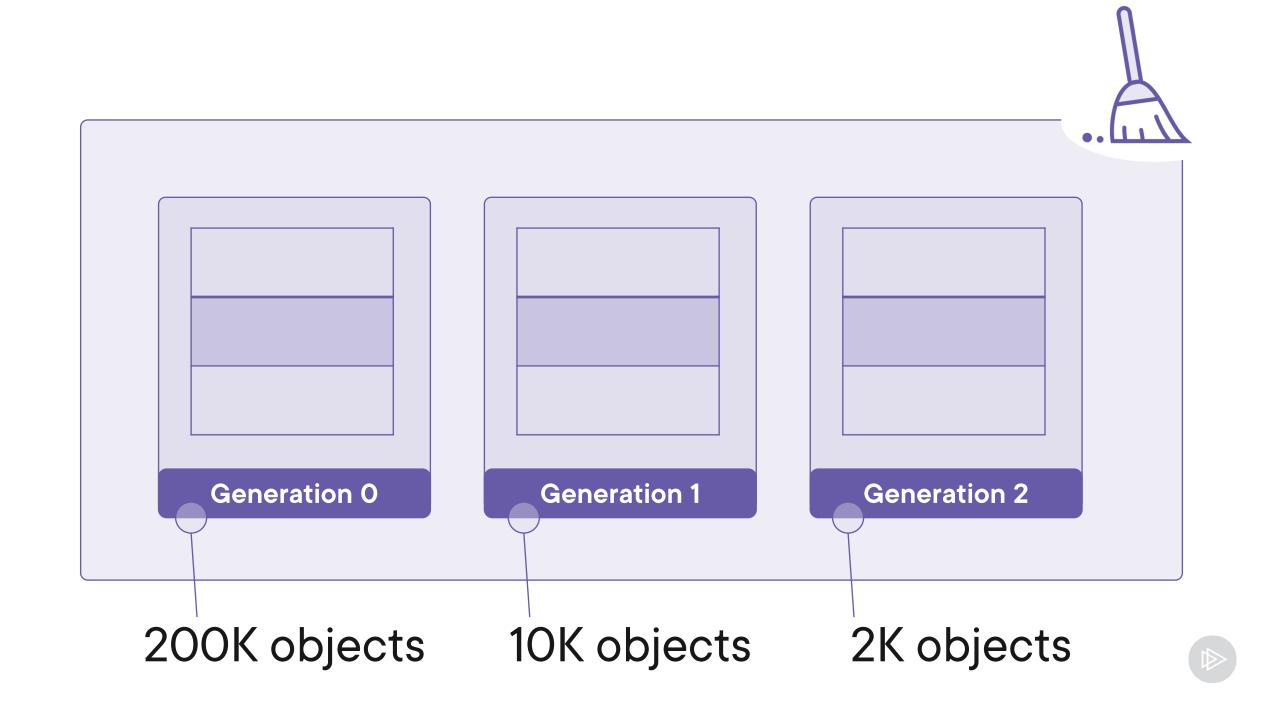


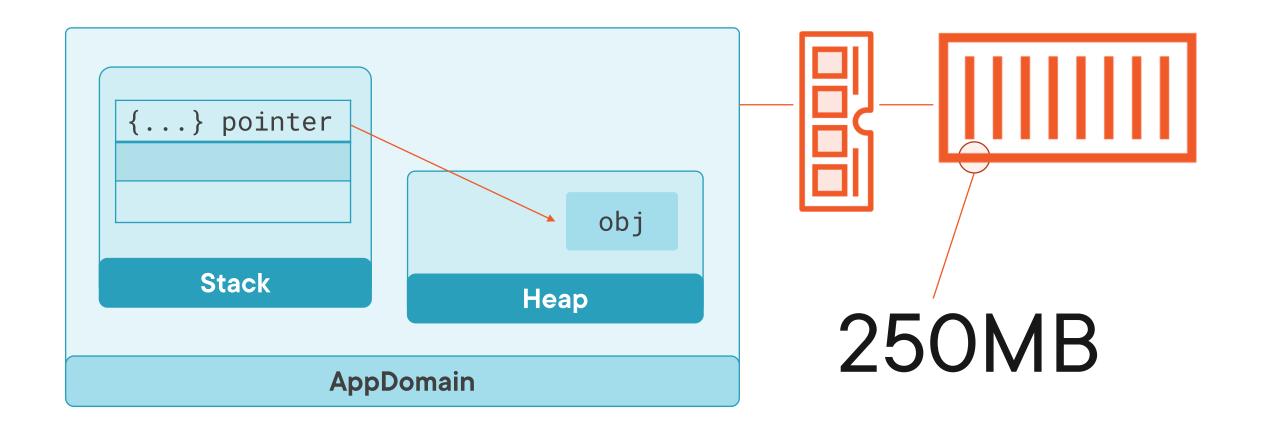












Disposing to Minimize GC Work

NotDisposing.cs

```
# object and it's graph gets cleaned
# up when the GC runs:
var s = new DatabaseState(_Config));
Console.WriteLine(s.GetDate());
```

Disposing.cs

```
using (var s = new DatabaseState(_Config))
{
   Console.WriteLine(s.GetDate());
}
# object and it's graph cleaned up now
```

Demo



Implementing IDisposable

- Using the dispose pattern
- Supporting subclasses
- Safely disposing objects



Demo



Implementing IDisposable

- Disposing local variables
- Profiling objects after disposing
- Reducing the GC workload



Disposing Local Variables

Program.cs

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

DatabaseState.cs

```
public class DatabaseState : IDisposable
{
   public void Dispose()
   {
      Dispose(true);
      GC.SuppressFinalize(this);
   }
```

```
protected void Dispose(bool disposing)
 if (_disposed)
    return;
 if (disposing)
   if (_connection != null)
     _connection.Dispose();
      _connection = null;
    _disposed = true;
```

◄ Dispose of disposable objects

Best Practice #2

If you use IDisposable objects as instance fields, implement IDisposable



```
protected SqlConnection _connection;
protected void Dispose(bool disposing)
 if (_disposed)
    return;
 if (disposing)
   if (_connection != null)
      _connection.Dispose();
      _connection = null;
    _disposed = true;
```

■ Local field is IDisposable

■ Check disposable object is live

■ Dispose and set to null

■ Allow multiple Dispose() calls

Best Practice #3

Allow Dispose() to be called multiple times and don't throw exceptions



```
public class ClassWithFinalizer : IDisposable
  ~ClassWithFinalizer() { /* ... */ }
                   obj
    Stack
                   Heap
          AppDomain
```

```
public class ClassWithFinalizer : IDisposable
  public void Dispose() { /* ... */ }
  obj
    Stack
                   Heap
          AppDomain
```

Demo



IDisposable and Finalizers

- Unmanaged local fields
- The protected Dispose() method
- Tracking disposal



Demo



IDisposable and Finalizers

- Implementing a finalizer
- Cleaning up all objects
- Profiling disposable objects

```
public class UnmanagedDatabaseState
                        : DatabaseState
 private SqlCommand _command;
 private IntPtr _unmanagedPointer;
 protected override
    void Dispose(bool disposing)
   if (disposing)
     if ( command != null)
         _command.Dispose();
         command = null;
   // clean up unmanaged resources
```

- **◄** Derived from an IDisposable class
- Additional disposable field
- **◄** Unmanaged resource

■ Override base method

■ Safely dispose

◄ Take care of unmanaged memory

```
public class UnmanagedDatabaseState
                         : DatabaseState
 protected override
     void Dispose(bool disposing)
   // clean up unmanaged resources
   if (_unmanagedPointer != IntPtr.Zero)
     Marshal FreeHGlobal
              ( unmanagedPointer);
      unmanagedPointer = IntPtr.Zero;
   // let base class clean up
    base.Dispose(disposing);
```

■ Base method override continued

■ Safely clean up

■ Ripple the dispose call

```
public class UnmanagedDatabaseState
                         : DatabaseState
 protected override
     void Dispose(bool disposing)
 ~UnmanagedDatabaseState()
   Dispose(false);
```

■ All cleanup in here

- Called by GC when object not disposed
- **◄** Clean up only unmanaged resources

Best Practice #4

Implement IDisposable to support disposing resources in a class hierarchy



Dispose Pattern

BaseClass.cs

```
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
protected virtual void Dispose(bool disposing)
{
    // dispose only *this* class's resources
}
```

DerivedClass.cs

```
protected override void Dispose(bool disposing)
{
    // dispose only *this* class's resources
}
```

Best Practice #5

If you use unmanaged resources, declare a finalizer which cleans them up



```
protected override void Dispose(bool disposing)
    if (disposing) { // clean up managed resources }
    // clean up unmanaged resources
    base.Dispose(disposing);
~UnmanagedDatabaseState()
    Dispose(false);
```

Cleaning Up Resources

Unmanaged: always. Managed: only if disposing.

Summary



Understanding the GC

- Memory management
- Minimal processing

Monitoring object allocation

- '000s of objects
- Grows when not disposing
- Flat when disposing

Implementing IDisposable

- Disposing resources safely
- Supporting inheritence
- Finalizers for unmanaged resources



Up Next:

What Happens If You Don't Dispose?

