Armen Melkumyan • 1st
Technical / Solutions Architect
6mo •

🔥 Boost Your App Performance: The True Cost of Heap Allocations

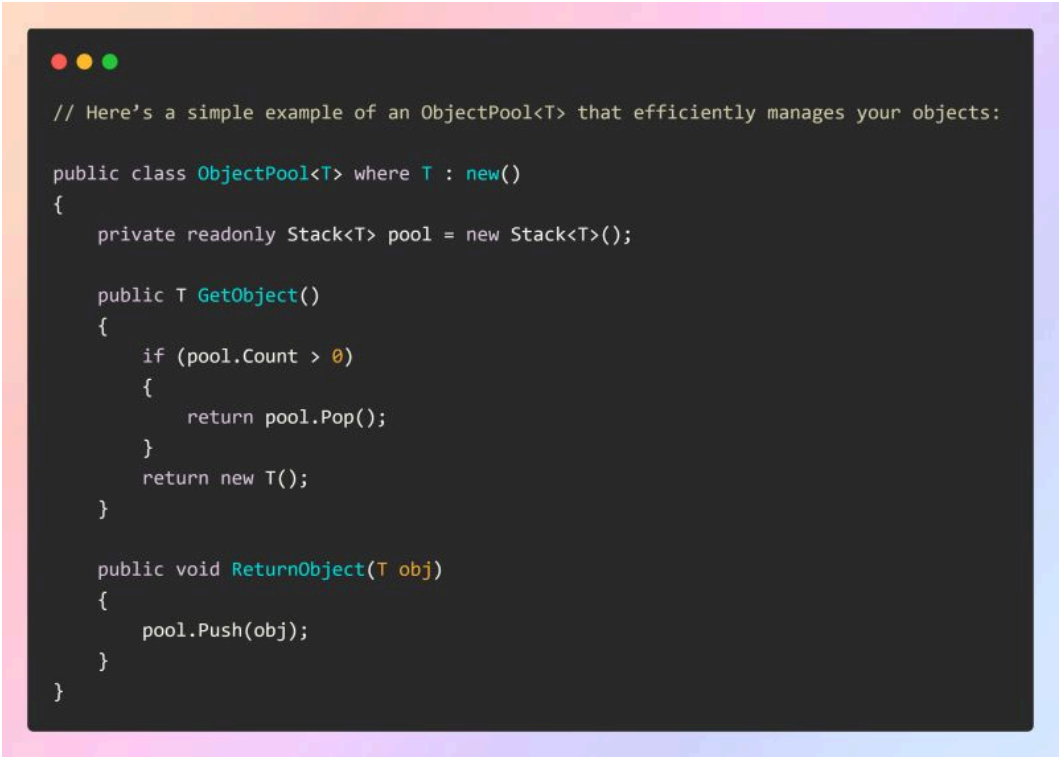
In high-performance applications, creating objects on the heap does more than just occupy memory—it adds overhead for the runtime to track these objects for garbage collection (GC). As the heap grows, GC cycles become more frequent, leading to GC pauses that can slow down your app. 🚫

💡 Solution: Enter Object Pooling! Reuse frequently created and destroyed objects to reduce heap allocations and keep that GC pressure down.

What happens under the hood:
ObjectPool<T> maintains a stack of reusable objects to avoid repetitive heap allocations.

Instead of letting objects be garbage collected when no longer needed, they're returned to the pool for reuse!
This is just a sneak peek from my upcoming book "High Performance .NET: Recipes and Thoughts" where we dive deep into optimizing .NET apps for speed, memory efficiency, and overall performance. Stay tuned!

#HighPerformanceDotNet #MemoryManagement #ObjectPooling #DotNetOptimizations #GarbageCollection #DotNet #CodingBestPractices #UpcomingBook 📘



```
// Here's a simple example of an ObjectPool<T> that efficiently manages your objects:

public class ObjectPool<T> where T : new()
{
    private readonly Stack<T> pool = new Stack<T>();

    public T GetObject()
    {
        if (pool.Count > 0)
        {
            return pool.Pop();
        }
        return new T();
    }

    public void ReturnObject(T obj)
    {
        pool.Push(obj);
    }
}
```

50 1 comment