Exercise 1: The Basic Eager Singleton

Goal: Create a ConfigurationManager that loads settings once and provides them globally.

- **Requirements:** * Use **Eager Initialization** (instantiate the object directly on the static field line).
    - The class must be sealed.
    - Add a property string AppName { get; set; }.
- **Success Criteria:** Prove that if you set AppName in one part of your code, it remains changed when accessed from another variable.

Exercise 2: The Double-Check Lock (Thread Safety)

Goal: Implement a thread-safe Logger class manually.

- **Requirements:**
    - Use a private static variable for the instance, initialized to null.
    - Create a private static readonly object to use as a sync root (the "padlock").
    - Implement the **Double-Check Locking** logic within the Instance property getter.
- **Success Criteria:** The code should only enter the lock block the very first time the logger is requested.

Exercise 3: The "Static Constructor" Approach

Goal: Use a static constructor to initialize a FileService.

- **Requirements:**
    - Instead of initializing the instance on the field line, use a static constructor: static FileService() { ... }.
    - Include a private instance constructor.
    - Add a method WriteToFile(string message) that simulates a file write.
- **Success Criteria:** Understand that the .NET Runtime guarantees the static constructor is called exactly once before any instance is created or any static member is referenced.

Exercise 4: The Hardware Interface (Resource Management)

Goal: Simulate a singleton for a hardware resource, like a PrinterSpooler.

- **Requirements:**
    - Use a manual Singleton (your choice of Eager or Thread-Safe).
    - Add an int _jobCount private field.
    - Add a method Print(string document) that increments and prints the _jobCount.
- **Success Criteria:** Create three different variables calling PrinterSpooler.Instance and ensure the _jobCount increments correctly across all of them (e.g., Job 1, Job 2, Job 3).

Exercise 5: The Singleton Registry (Advanced)

Goal: Create a Singleton that manages a Dictionary<string, string> of session data.

- **Requirements:**
    - Ensure the class cannot be inherited.
    - Make the constructor private.
    - Initialize the dictionary inside the class.
    - Prevent the dictionary itself from being overwritten from the outside (make it private or readonly).
- **Success Criteria:** Demonstrate that this "Session Manager" can store a user's login state globally without using a database or DI container.