Day 43/60 - What is the volatile Keyword in C#?

In multithreading, data consistency is critical when multiple threads access shared variables. The volatile keyword ensures that a variable is always read from and written to main memory, preventing compiler optimizations that could cause stale values.

When to Use volatile?
- When multiple threads are reading/writing a shared field.
- When you need to prevent caching or reordering optimizations.

Example: Without volatile (May Cause Issues)

```
class Example
{
 private static bool _isRunning = true;

 static void Main()
 {
 new Thread(() =>
 {
 while (_isRunning) { } // May never see the updated value
 }).Start();

 Thread.Sleep(1000);
 _isRunning = false; // This update might not be seen by the loop
 }
}
```

Here, the CPU may cache _isRunning, causing the loop to never exit.

Example: Using volatile (Ensures Visibility)

```
class Example
{
 private static volatile bool _isRunning = true;

 static void Main()
 {
 new Thread(() =>
 {
 while (_isRunning) { } // Always reads fresh value
 }).Start();

 Thread.Sleep(1000);
 _isRunning = false; // Change is immediately visible to all threads
 }
}
```

Now, _isRunning is always fetched from main memory, ensuring thread visibility.

Limitations of volatile
- Works only with fields, not local variables.
- Doesn’t provide atomicity (use lock or Interlocked for operations like x++).
- Not needed for reference types, as their reference assignment is already atomic.

Use volatile wisely to avoid unpredictable threading issues in .NET!

#dotnet #csharp #multithreading #volatile #performance