Day 49/60 - Understanding readonly vs const vs static readonly in C#

Choosing between const, readonly, and static readonly can impact your code's performance, maintainability, and flexibility. Here's how they differ:

const (Compile-Time Constant)

Must be assigned at declaration.
Value is fixed at compile time and cannot be changed.
Stored in assembly metadata, not in memory.

Example:

```
public class Example
{
 public const double Pi = 3.14159; // Value is fixed at compile time
}
```

Limitations: Cannot change the value at runtime.

readonly (Runtime Constant, Instance-Level)

Value can be set only in the constructor.
Evaluated at runtime, allowing different instances to have different values.

Example:

```
public class Example
{
 public readonly int MaxValue;

 public Example(int value)
 {
 MaxValue = value; // Value assigned at runtime
 }
}
```

Great for values that depend on constructor parameters.

static readonly (Runtime Constant, Shared Across Instances)

Similar to readonly, but applies to the entire class instead of each instance.
Can be initialized in a static constructor.

Example:

```
public class Example
{
 public static readonly int MaxLimit;

 static Example()
 {
 MaxLimit = 100; // Can only be set in static constructor
 }
}
```

Best for configuration values that should be shared across all instances.

When to Use What?

Use const for fixed values like Pi.
Use readonly for values that can change at runtime but should remain constant per instance.
Use static readonly for values that should remain constant across all instances.

Using the right modifier improves performance, maintainability, and clarity in your .NET applications!

#dotnet #csharp #constants #readonly #staticreadonly