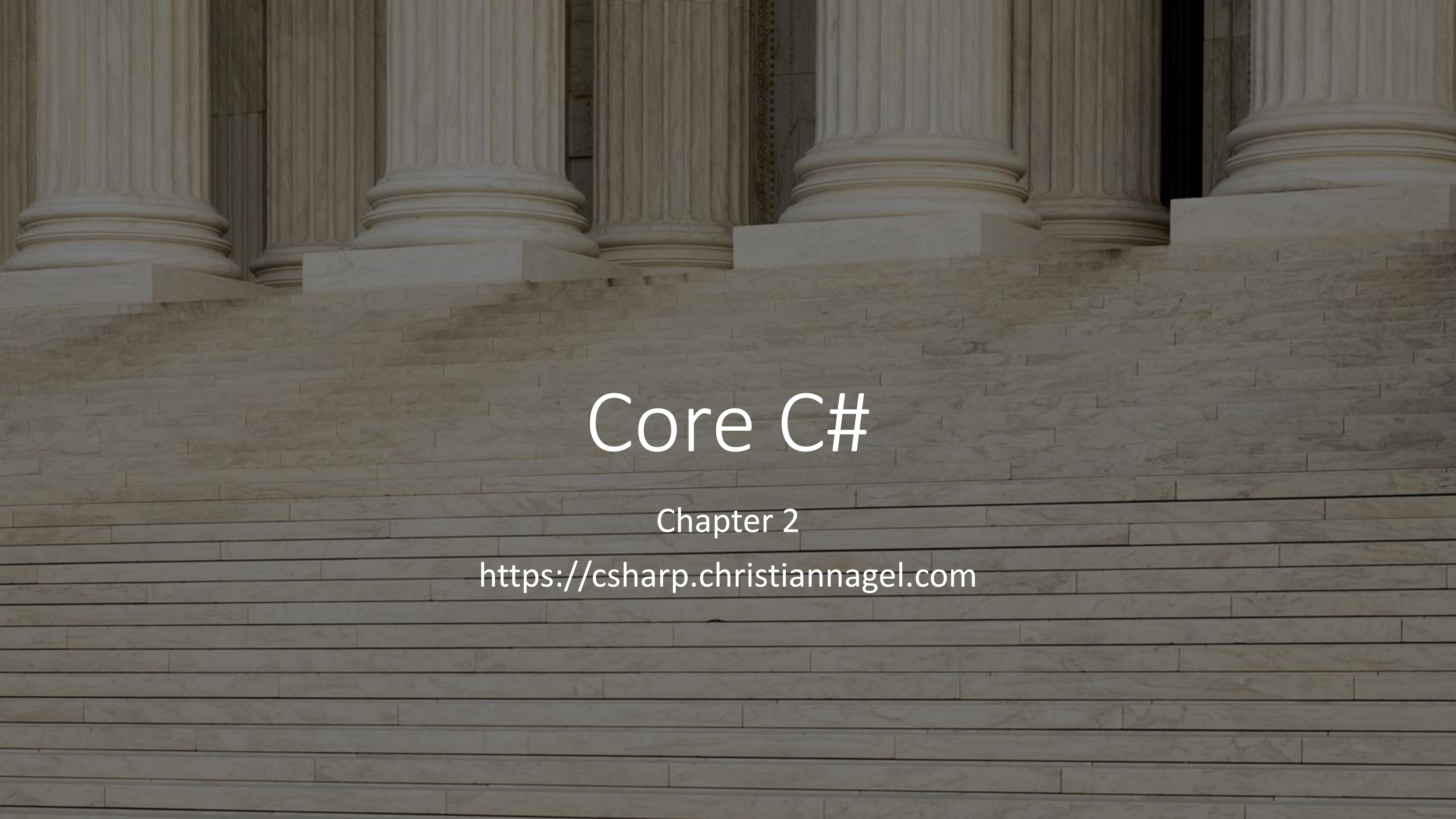


Core C#

Chapter 2

https://csharp.christiannagel.com

Topics

- Fundamentals of C#
- Nullable Types
- Predefined Types
- Controlling Program Flow
- Namespaces
- Strings
- Comments
- Preprocessor directives
- Programming Guidelines

Fundamentals

Variables

- Declare and initialize Variables

```
int i1 = 1; // declaration and initialization
var i2 = 2; // type inference

string s1 = new string("Hello, World!");
string s2 = "Hello, World!";
var s3 = "Hello, World!";
string s4 = new("Hello, World!");
```

Command-Line Arguments

- args Variable with top-level statements
- Main method with string array

```
using System;

foreach (var arg in args)
{
  Console.WriteLine(arg);
}
```

Variable Scope

- Region of code from where the variable can be accessed
- A field (member variable) is in scope as long as the class is in scope
- A local variable is in scope until a closing brace indicates the end of the block statement
- A local variable in for/while/… is in scope in the bode of the loop

Constants

- Value cannot be changed
- Replaced by the compiler
- Must be initialized at declaration
- const int a = 100;

Nullable Value Types

- int? x = null;
- Compiler uses Nullable<int>
- HasValue
- Value

Nullable Reference Types

- Enable Nullable in the project <Nullable>
- Nullable in the source code #nullable enable/disable/restore

```
string s1 = GetAString();

string? s2 = GetAStringOrNull();

if (s2 is not null)
{
  s1 = s2;
}
```

Predefined Types

- Integer Types
- Binary Values
- Floating-Point Types
- Boolean Type
- Character Type

Integer Types

- Integer Types
  - short (Int16)
  - int (Int32)
  - long (Int64)
- Digit Separators
  - long l3 = 0x_1234_5678_9abc;
- Binary Values
  - uint binary1 = 0b_1111_1110_1101_1100_1010_1001_1000;

Floating-Point Types

C# Keyword	.NET Type	Description	Significand bit	Exponent bit
	Half	16-bit single-precision	10	5
float	Single	32-bit single-precision	23	8
double	Double	64-bit double-precision	52	11
decimal	Decimal	bits[0] .. bits[2] – 96-bits integer number bits[3] scale factor and sign		

Boolean Type

- bool b1 = true;
- true / false

Character Type

- Two-type characters
- char maps to System.Char
- Single quotations create a char

Number Literals

- U unsigned int
- L long
- UL unsigned long
- F float
- M decimal
- 0x hex number (prefix)
- 0b binary number (prefix)

Control Program Flow

if Statement

- if / else if / else

```
if (string.IsNullOrEmpty(input))
{
  Console.WriteLine("You typed in an empty string.");
}
else if (input?.Length < 5)
{
  Console.WriteLine("The string had less than 5 characters.");
}
else
{
  Console.WriteLine("Read any other string");
}
```

Pattern Matching with the is operator

- Const Pattern
- Type Pattern

```
if (o is null) throw new ArgumentNullException(nameof(o));
else if (o is Book b)
{
  Console.WriteLine($"received a book: {b.Title}");
}
```

switch Statement

```
void SwitchSample(int x)
{
  switch (x)
  {
    case 1:
      Console.WriteLine("integerA = 1");
      break;
    case 2:
      Console.WriteLine("integerA = 2");
      break;
    case 3:
      Console.WriteLine("integerA = 3");
      break;
    default:
      Console.WriteLine("integerA is not 1, 2, or 3");
      break;
  }
}
```

switch Statement with Pattern Matching

```
void SwitchWithPatternMatching(object o)
{
  switch (o)
  {
    case null:
      Console.WriteLine("const pattern with null");
      break;
    case int i when i > 42
      Console.WriteLine("type pattern with when and a relational pattern");
      break;
    case int:
      Console.WriteLine("type pattern with an int");
      break;
    case Book b:
      Console.WriteLine($"type pattern with a Book {b.Title}");
      break;
    default:
      break;
  }
}
```

switch expression

```
TrafficLight NextLightClassic(TrafficLight light)
{
 switch (light)
 {
  case TrafficLight.Green:
   return TrafficLight.Amber;
  case TrafficLight.Amber:
   return TrafficLight.Red;
  case TrafficLight.Red:
   return TrafficLight.Green;
  default:
    throw new InvalidOperationException();
 }
}
```

```
TrafficLight NextLight(TrafficLight light) =>
 light switch
 {
  TrafficLight.Green => TrafficLight.Amber,
  TrafficLight.Amber => TrafficLight.Red,
  TrafficLight.Red => TrafficLight.Green,
  _ => throw new InvalidOperationException()
 };
```

for loop

```
for (int i = 0; i < 100; i++)
{
  Console.WriteLine(i);
}
```

```
for (int i = 0; i < 100; i += 10)
{
  // This loop iterates through columns
  for (int j = i; j < i + 10; j++)
  {
    Console.Write($" {j}");
  }
  Console.WriteLine();
}
```

while loop

```
bool condition = false;
while (!condition)
{
  // This loop spins until the condition is true.
  DoSomeWork();
  condition = CheckCondition(); // assume CheckCondition() returns a bool
}
```

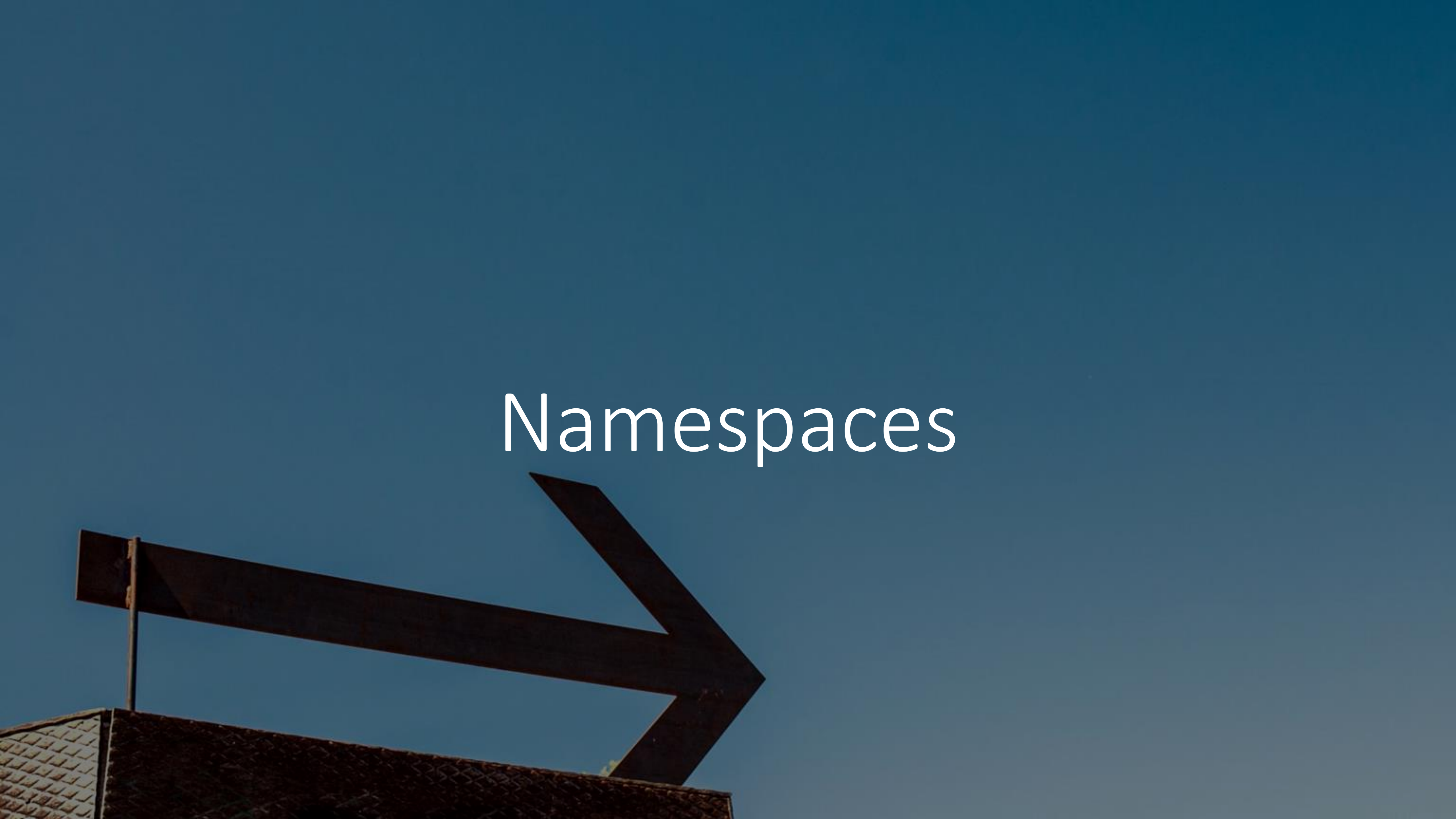
do-while loop

```
bool condition;
do
{
  // This loop will at least execute once, even if the condition is false.
  MustBeCalledAtLeastOnce();
  condition = CheckCondition();
} while (condition);
```

foreach loop

```
foreach (int temp in arrayOfInts)
{
  Console.WriteLine(temp);
}
```

Namespaces



Namespaces

- Hierarchical organization of types

```
namespace Wrox
{
  namespace ProCSharp
  {
    namespace CoreCSharp
    {
      public class Sample
      {
      }
    }
  }
}
```

```
// dotted-notation
namespace Wrox.ProCSharp.CoreCSharp
{
  public class Sample
  {
  }
}
```

Using Directive

- Import the namespace

```
using Wrox.ProCSharp.CoreCSharp;

Sample sample1 = new();
```

- Namespace Alias

```
using TimersTimer = System.Timers.Timer;
using WebTimer = System.Web.UI.Timer;
```

- Using Static

```
using static System.Console;

WriteLine("Hello, World!");
```

Working with Strings

String Concatenation

- Creates temporary strings

```
string s1 = "Hello";
string s2 = "World";
string s3 = s1 + "  " + s2;
```

StringBuilder

- Uses a buffer
- Resize dynamically

```
void UsingStringBuilder()
{
  StringBuilder sb = new("the quick");
  sb.Append(' ');
  sb.Append("brown fox jumped over ");
  sb.Append("the lazy dogs 1234567890 times");
  string s = sb.ToString();
  Console.WriteLine(s);
}
```

String Interpolation

- Uses a buffer
- Resize dynamically

```
void UsingStringBuilder()
{
  StringBuilder sb = new("the quick");
  sb.Append(' ');
  sb.Append("brown fox jumped over ");
  sb.Append("the lazy dogs 1234567890 times");
  string s = sb.ToString();
  Console.WriteLine(s);
}
```

FormattableString

- Format, ArgumentCount properties, GetArgument method

```
void UsingFormattableString()
{
  int x = 3, y = 4;
  FormattableString s = $"The result of {x} + {y} is {x + y}";
  Console.WriteLine($"format: {s.Format}");
  for (int i = 0; i < s.ArgumentCount; i++)
  {
    Console.WriteLine($"argument: {i}:{s.GetArgument(i)}");
  }
  Console.WriteLine();
}
```

String Formats

- number, date, time formats

```
void UseStringFormat()
{
  DateTime day = new(2025, 2, 14);
  Console.WriteLine($"{day:D}");
  Console.WriteLine($"{day:d}");

  int i = 2477;
  Console.WriteLine($"{i:n} {i:e} {i:x} {i:c}");

  double d = 3.1415;
  Console.WriteLine($"{d:###.###}");
  Console.WriteLine($"{d:000.000}");
  Console.WriteLine();
}
```

Verbatim Strings

- @ prefix
- No escape characters needed

```
with the @ character:
string s = @"a tab: \t, a carriage return: \r, a newline: \n";
Console.WriteLine(s);
```

Ranges with Strings

- Range operator ..
- Hat operator ^

```
void RangesWithStrings()
{
  string s = "The quick brown fox jumped over the lazy dogs down " +
    "1234567890 times";
  string the = s[..3];
  string quick = s[4..9];
  string times = s[^5..^0];
  Console.WriteLine(the);
  Console.WriteLine(quick);
  Console.WriteLine(times);
  Console.WriteLine();
}
```

More…

Comments

- // one-line comments
- /* multi-line comments */
- /// <Summary>XML Documentation</Summary>

Preprocessor Directives

- #define #undef
- #if #elif #else #endif
- #warning #error
- #region #endregion
- #pragma (suppress/restore compiler warnings)
- #nullable

C# Programming Guidelines

- Identifiers
  - begin with letter or underscore
  - can’t use C# keywords
- Naming conventions
  - Pascal casing with namespaces, types, properties
  - Camel casing with fields, variables

Summary

- Variables
- Types
- Nullable Types
- Control Flow
- Strings