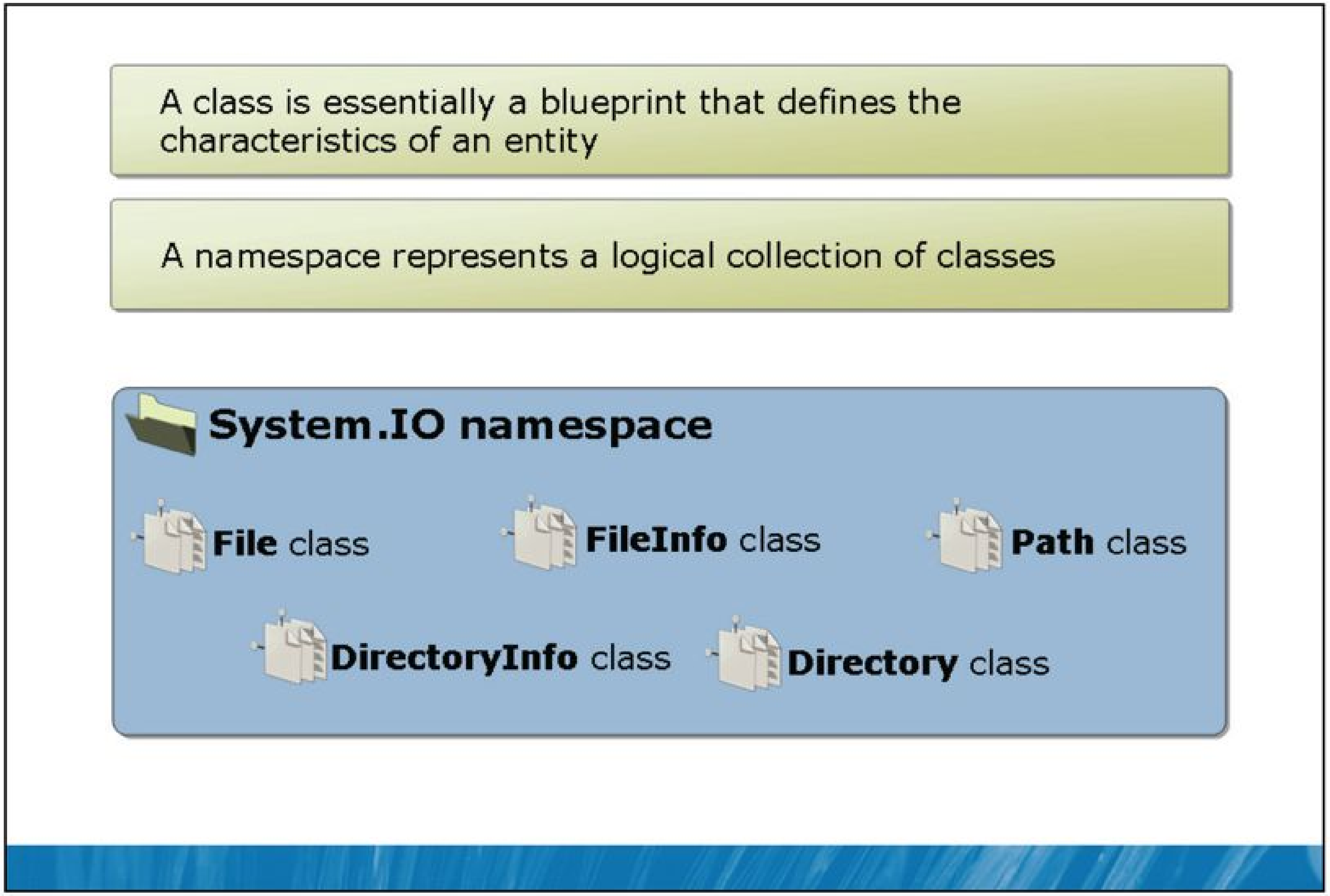
Microsoft® Official Course

Module02

C# Fundamentals

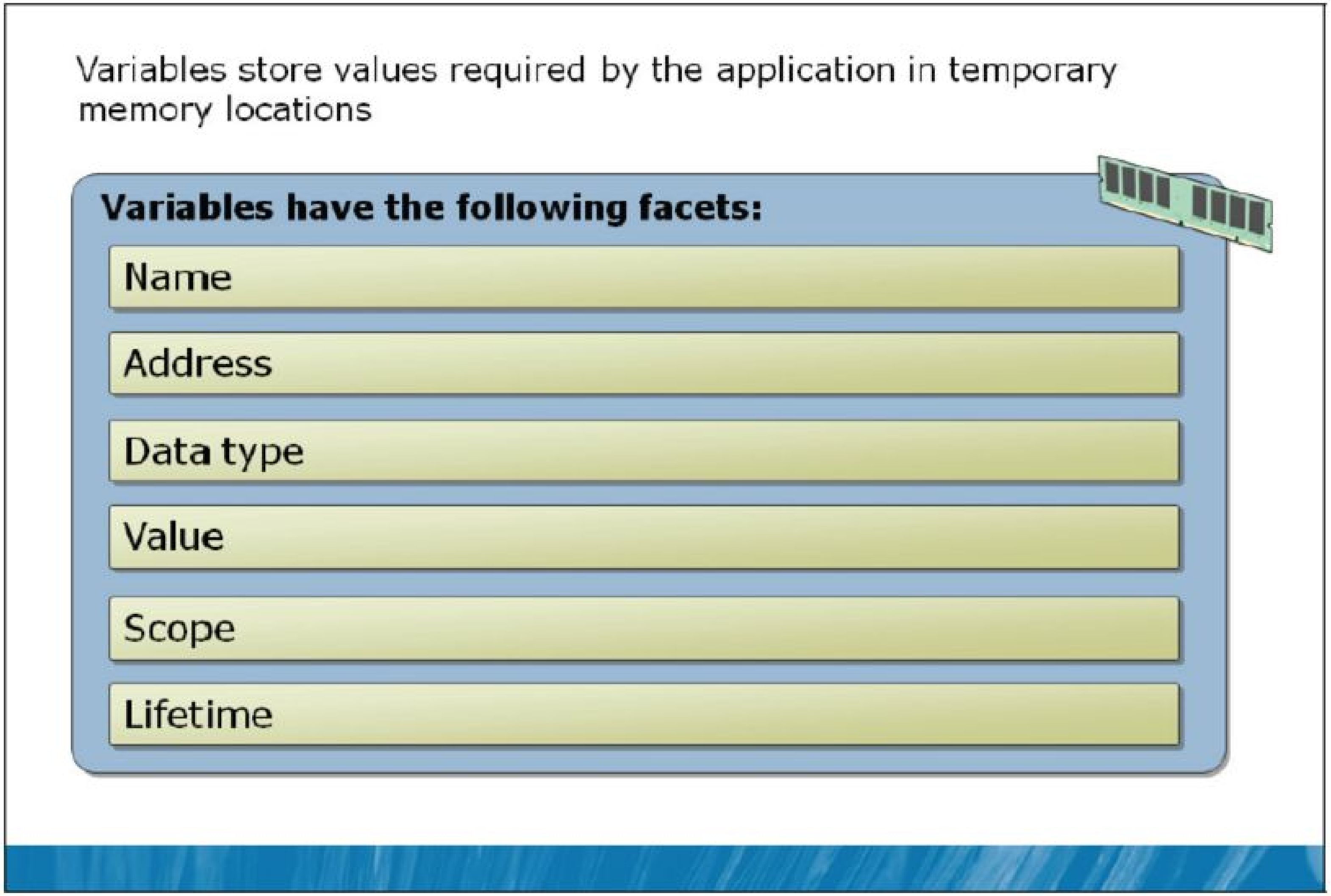
What Are Classes and Namespaces?



A class is essentially a blueprint that defines the characteristics of an entity

A namespace represents a logical collection of classes

What Are Variables?

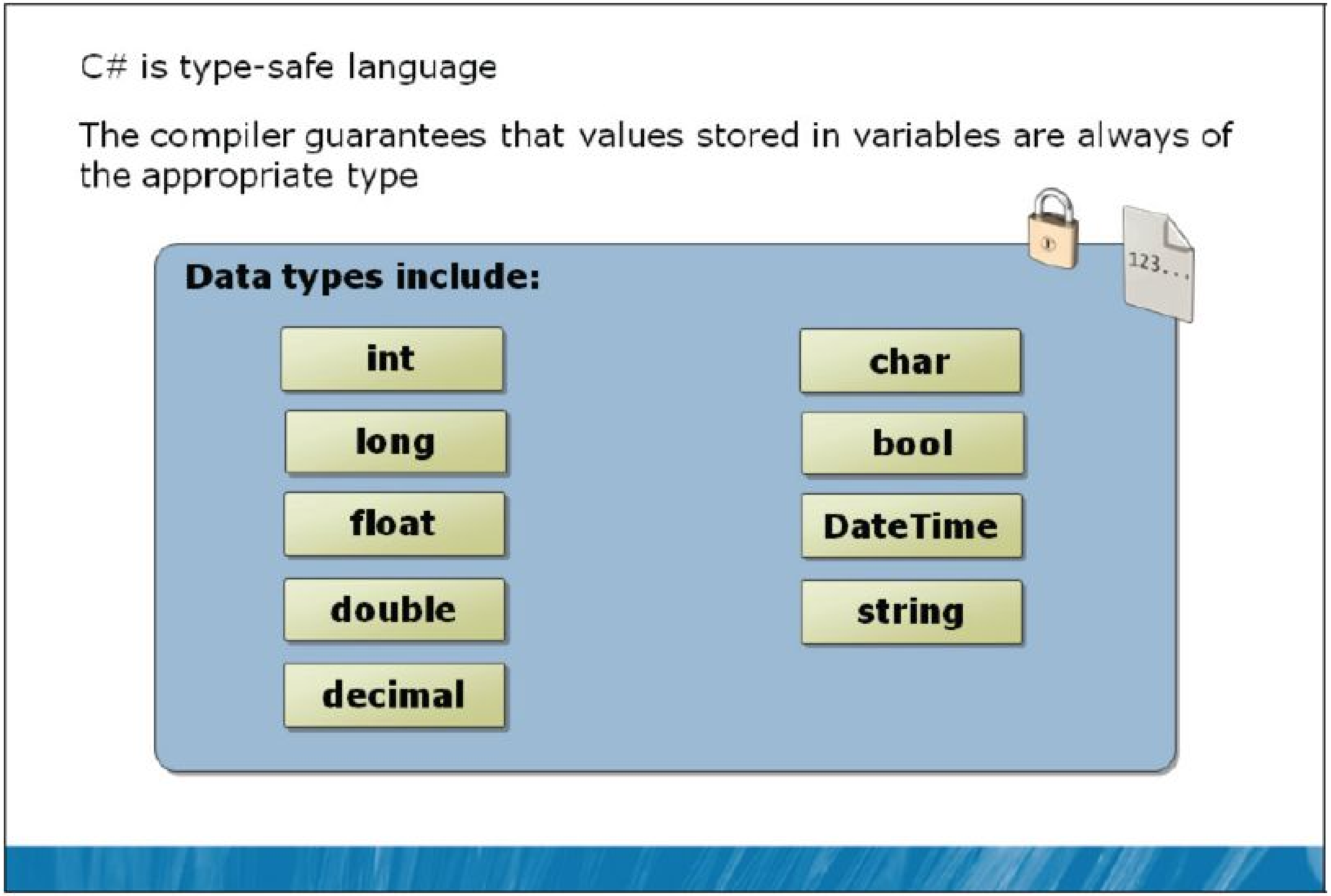


Variables store values required by the application in temporary memory locations

Variables have the following facets:

- Name
- Address
- Data type
- Value
- Scope
- Lifetime

What Are Data Types?



C# is type-safe language

The compiler guarantees that values stored in variables are always of the appropriate type

Data types include:

int	char
long	bool
float	DateTime
double	string
decimal	

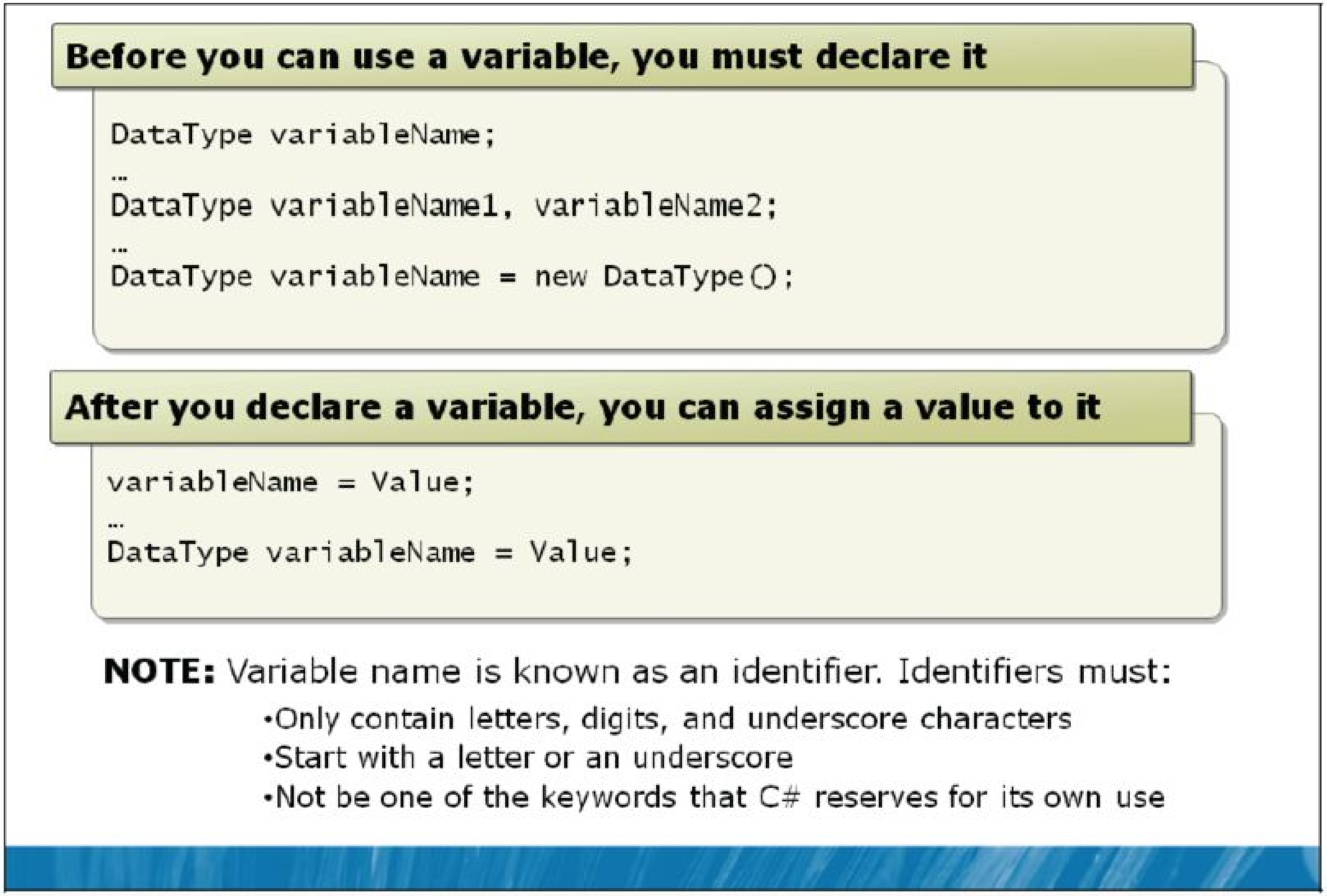
Integer

TYPE	ALIAS FOR	ALLOWED VALUES
`sbyte`	`System.SByte`	Integer between –128 and 127
`byte`	`System.Byte`	Integer between 0 and 255
`short`	`System.Int16`	Integer between –32768 and 32767
`ushort`	`System.UInt16`	Integer between 0 and 65535
`int`	`System.Int32`	Integer between –2147483648 and 2147483647
`uint`	`System.UInt32`	Integer between 0 and 4294967295
`long`	`System.Int64`	Integer between –9223372036854775808 and 9223372036854775807
`ulong`	`System.UInt64`	Integer between 0 and 18446744073709551615

Declaring Variables

- A **variable declaration** includes:
  - The first character of a variable name must be either a letter, an underscore character (_), or the at symbol (@).
  - Subsequent characters may be letters, underscore characters, or numbers.

Declaring and Assigning Variables Key



Before you can use a variable, you must declare it

```
DataType variableName;
…
DataType variableName1, variableName2;
…
DataType variableName = new DataType();
```

After you declare a variable, you can assign a value to it

```
variableName = Value;
…
DataType variableName = Value;
```

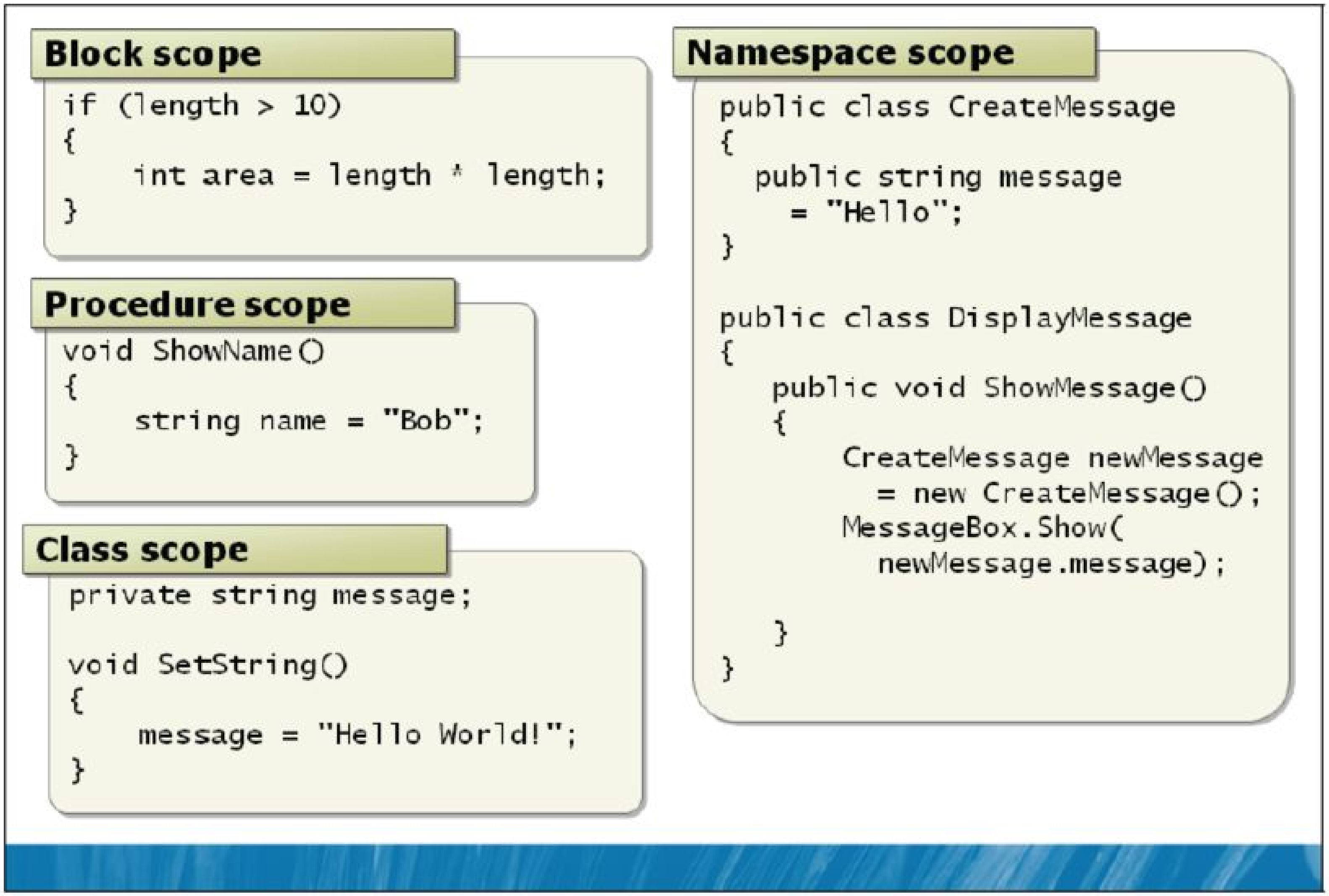
NOTE: Variable name is known as an identifier. Identifiers must:

- Only contain letters, digits, and underscore characters
- Start with a letter or an underscore
- Not be one of the keywords that C# reserves for its own use

Implicitly Typed Variables

- When you declare variables, you can also use the var keyword instead of specifying an explicit data type such as int or string.
- You must initialize a variable that is defined in this way when it is defined.
- var price = 20;

What Is Variable Scope?



Block scope

```
if (length > 10)
{
      int area = length * length;
}
```

Procedure scope

```
void ShowName()
{
      string name = "Bob";
}
```

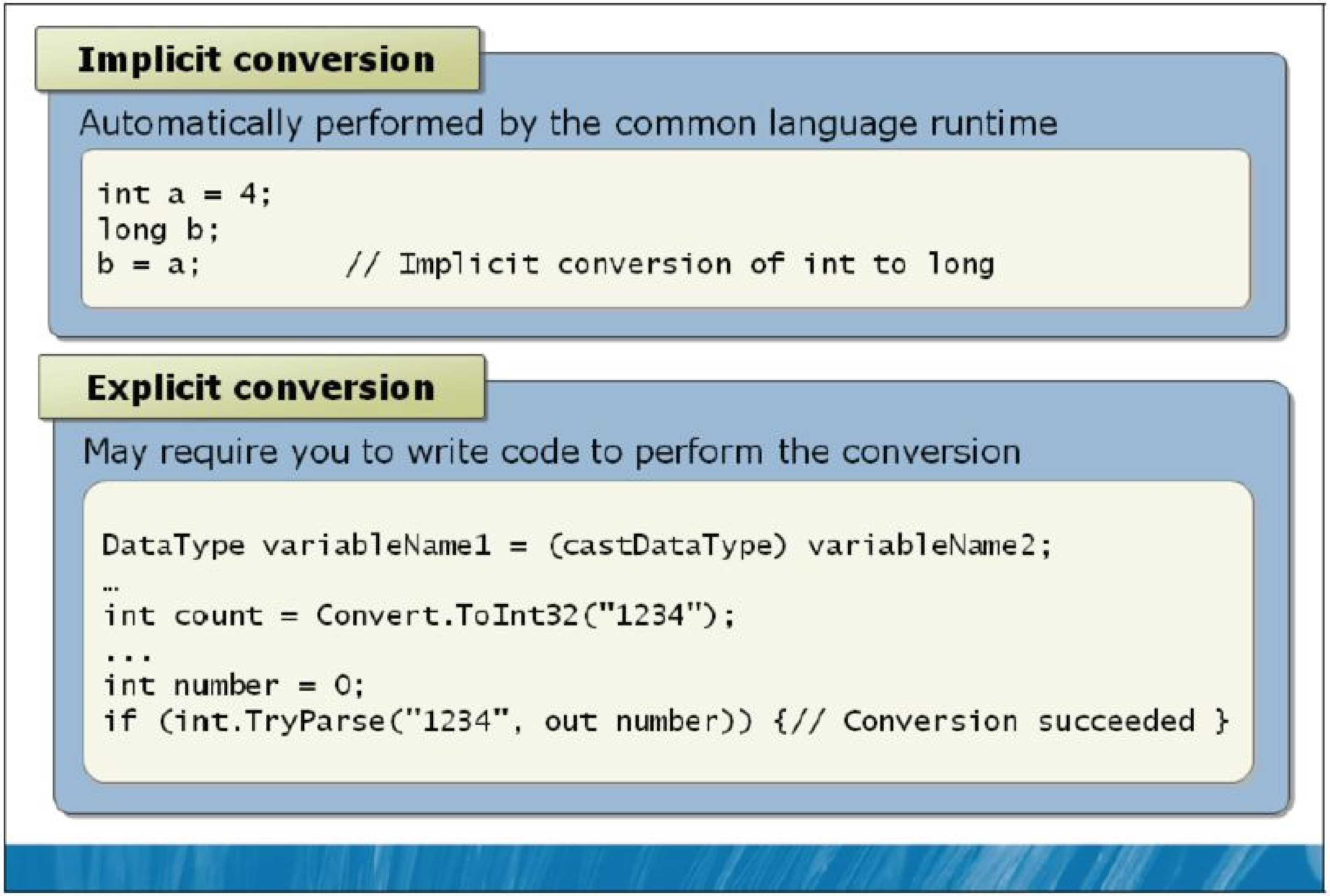
Class scope

```
private string message;

void SetString()
{
      message = "Hello World!";
}
```

Namespace scope

```
public class CreateMessage
{
   public string message
      = "Hello";
}

public class DisplayMessage
{
    public void ShowMessage()
    {
          CreateMessage newMessage
             = new CreateMessage();
          MessageBox.Show(
             newMessage.message);

    }
}
```

Converting a Value to a Different Data Type



Implicit conversion

Automatically performed by the common language runtime

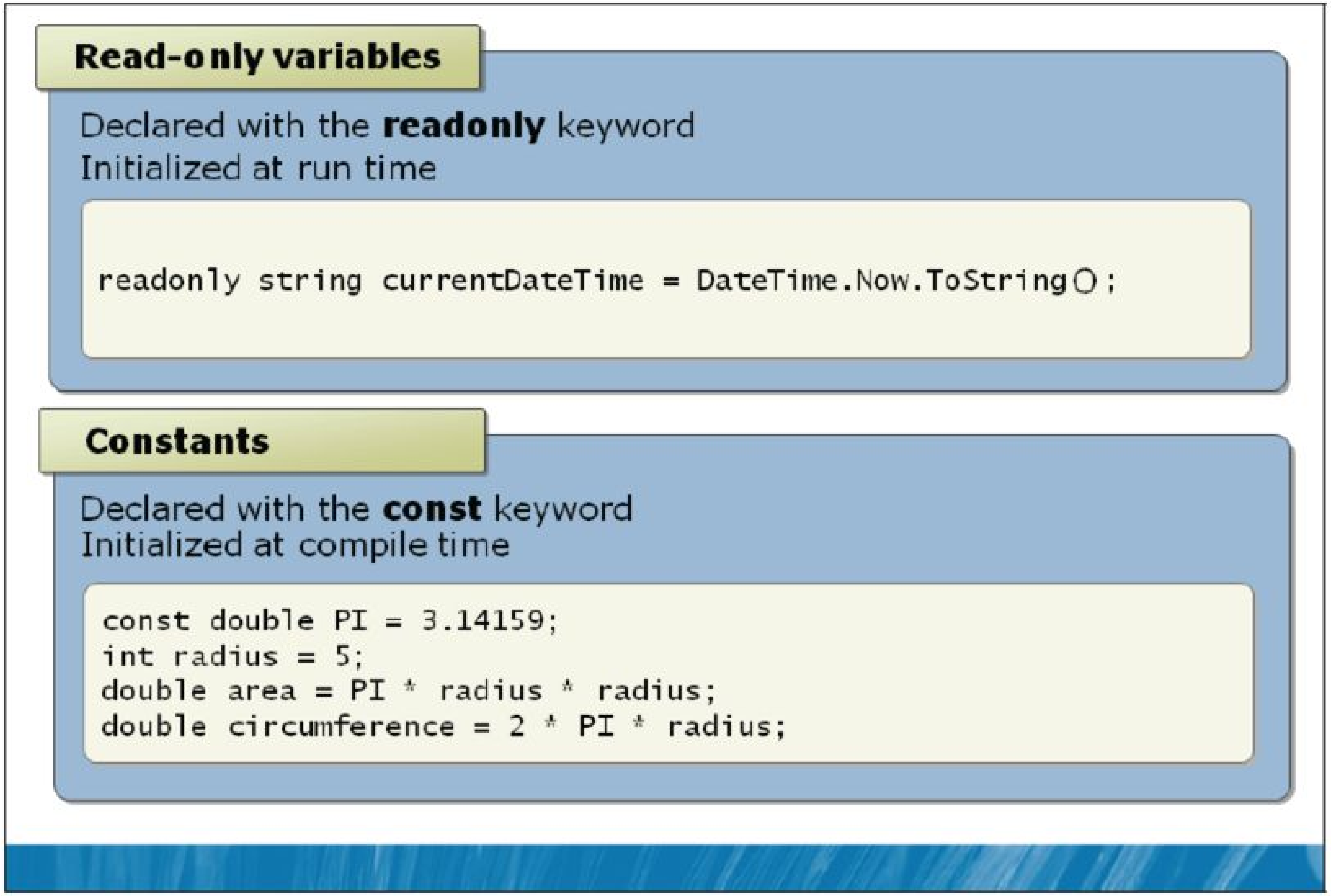
```
int a = 4;
long b;
b = a;          // Implicit conversion of int to long
```

Explicit conversion

May require you to write code to perform the conversion

```
DataType variableName1 = (castDataType) variableName2;
…
int count = Convert.ToInt32("1234");
...
int number = 0;
if (int.TryParse("1234", out number)) {// Conversion succeeded }
```

Read-Only Variables and Constants



Read-only variables

Declared with the **readonly** keyword
Initialized at run time

```
readonly string currentDateTime = DateTime.Now.ToString();
```

Constants

Declared with the **const** keyword
Initialized at compile time

```
const double PI = 3.14159;
int radius = 5;
double area = PI * radius * radius;
double circumference = 2 * PI * radius;
```

Using Expressions and Operators

Expressions are the fundamental constructs that you use to evaluate and manipulate data

```
a + 1

(a + b) / 2

"Answer: " + c.ToString()

b * System.Math.Tan(theta)
```

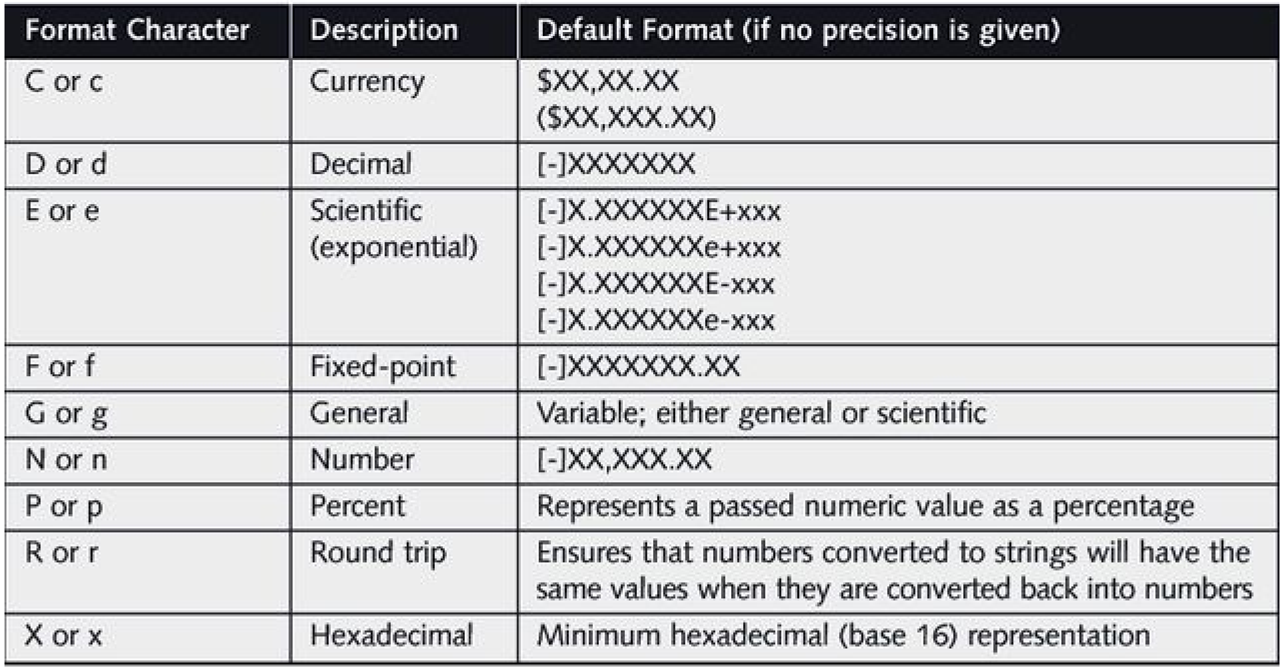
Displaying Variable Values

- The number within the curly braces in the format string must be less than the number of values you list after the format string
- You do not have to use the positions in order:
  - Console.WriteLine(“The money is {0} . ${0} is a lot for my age which is {1}”, someMoney, myAge);

Using Floating-Point Data Types

- A **floating-point** number is one that contains decimal positions
- C# supports three floating-point data types: float, double, and decimal
- A **double** can hold 15 to 16 **significant digits of accuracy**
- Floating-point numbers are double by default

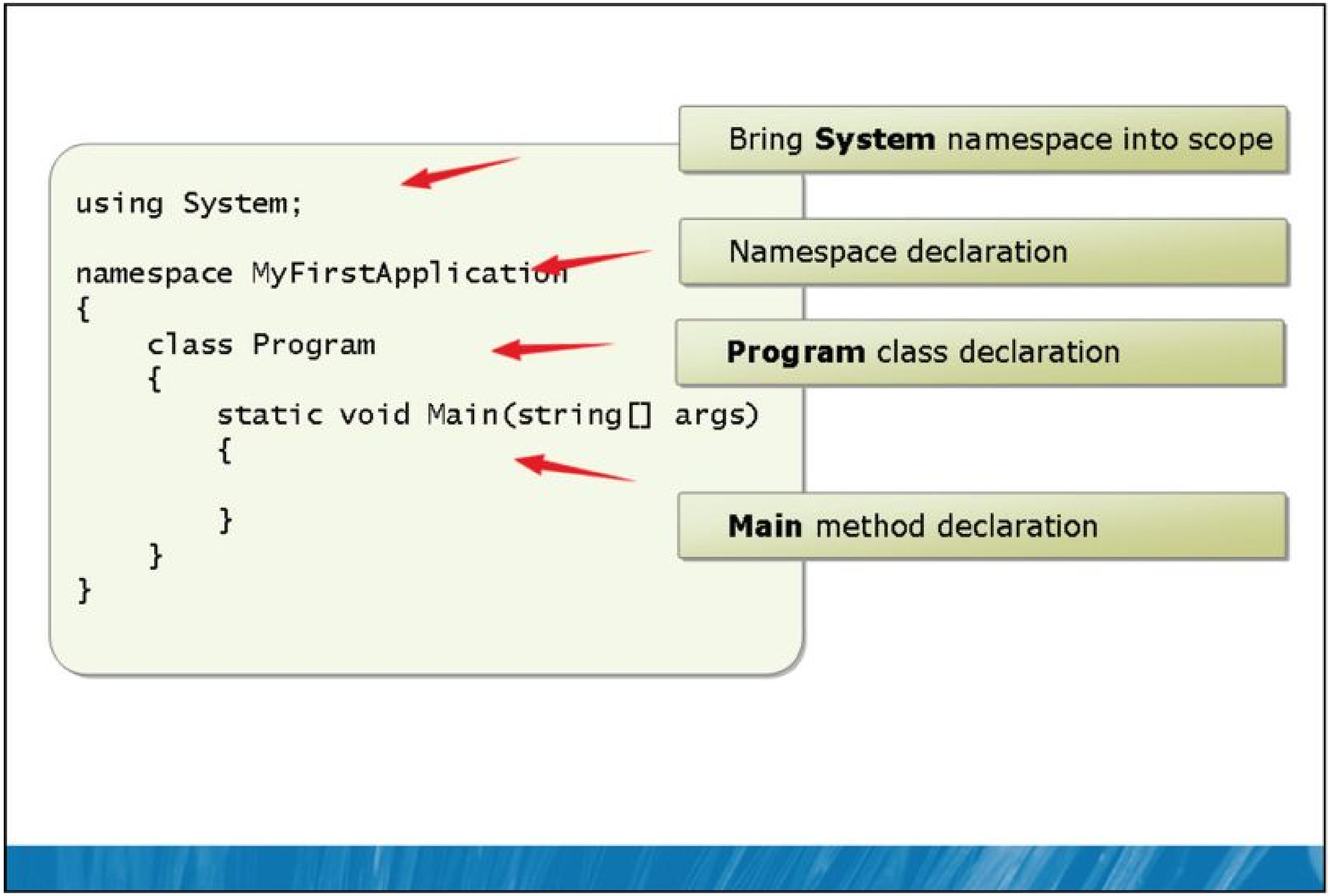
Formatting Floating-Point Values



Format Character	Description	Default Format (if no precision is given)
C or c	Currency	$XX,XX.XX ($XX,XXX.XX)
D or d	Decimal	[-]XXXXXXX
E or e	Scientific (exponential)	[-]X.XXXXXXE+xxx [-]X.XXXXXXe+xxx [-]X.XXXXXXE-xxx [-]X.XXXXXXe-xxx
F or f	Fixed-point	[-]XXXXXXX.XX
G or g	General	Variable; either general or scientific
N or n	Number	[-]XX,XXX.XX
P or p	Percent	Represents a passed numeric value as a percentage
R or r	Round trip	Ensures that numbers converted to strings will have the same values when they are converted back into numbers
X or x	Hexadecimal	Minimum hexadecimal (base 16) representation

```
Console.WriteLine(”Money {0:c}",4321.2);
```

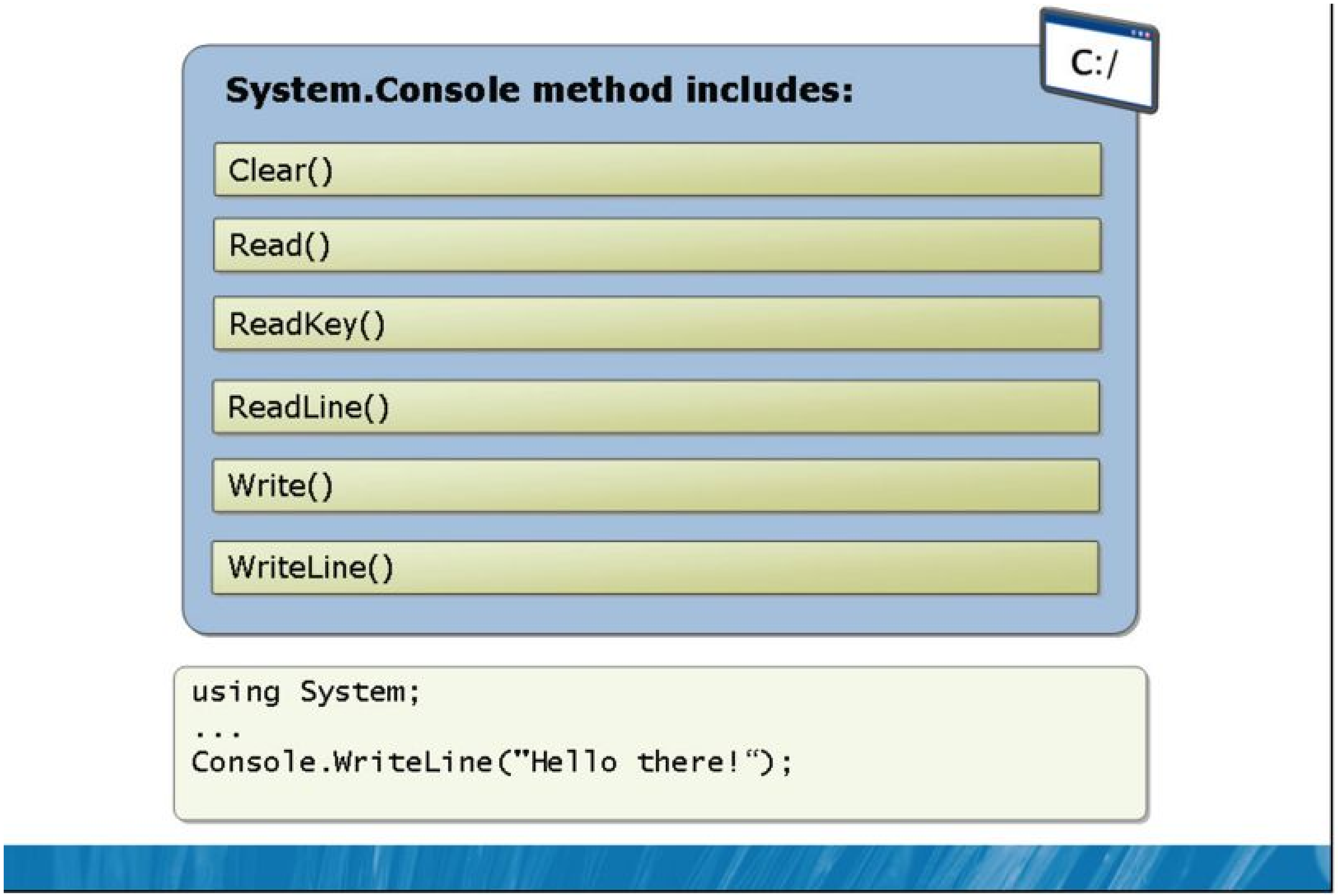
The Structure of a Console Application Key



```
using System;

namespace MyFirstApplication
{
    class Program
    {
        static void Main(string[] args)
        {

        }
    }
}
```

Bring **System** namespace into scope
Namespace declaration
Program class declaration
Main method declaration

Performing Input and Output by Using a Console Application

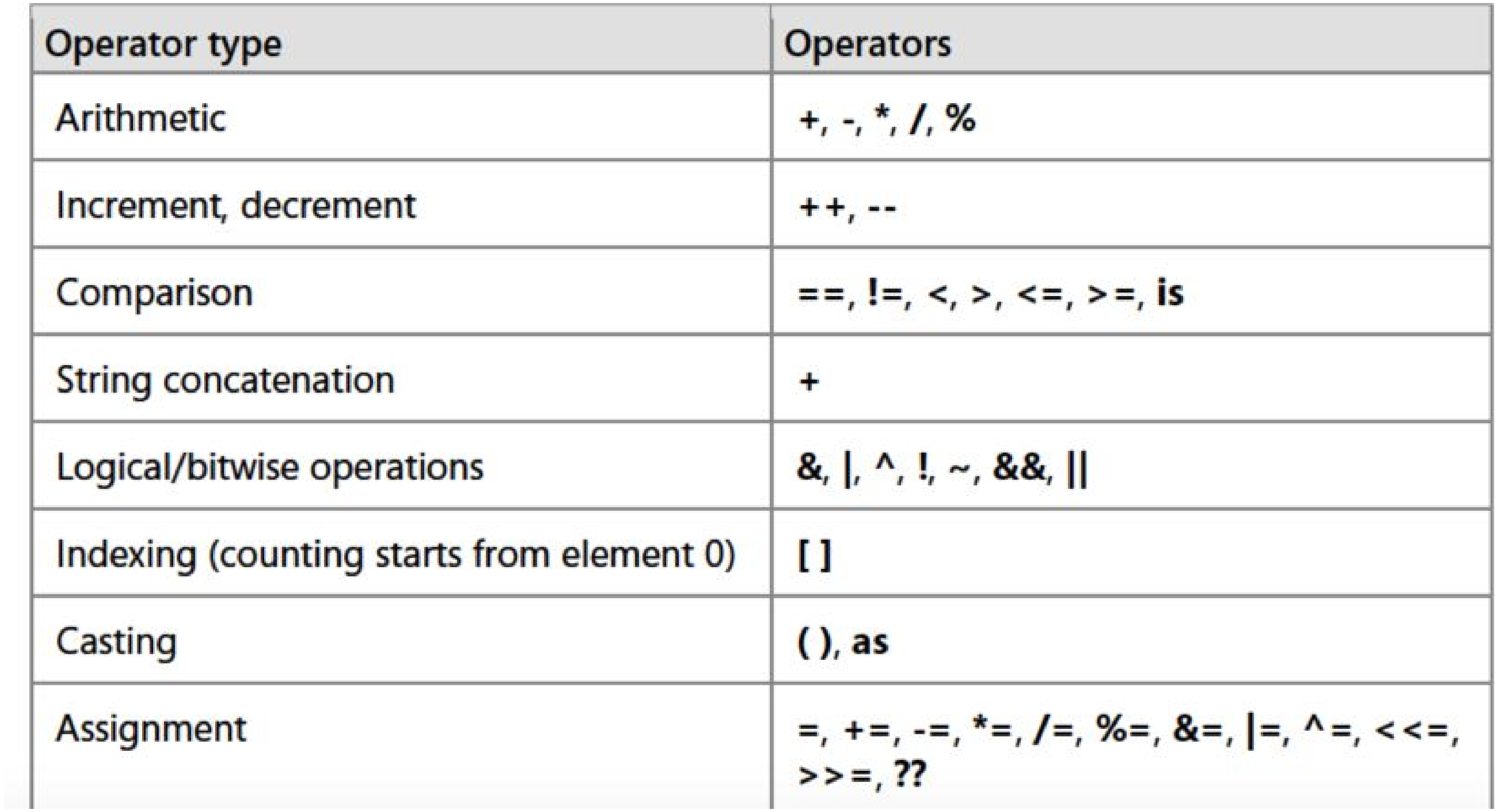


System.Console method includes:

- Clear()
- Read()
- ReadKey()
- ReadLine()
- Write()
- WriteLine()

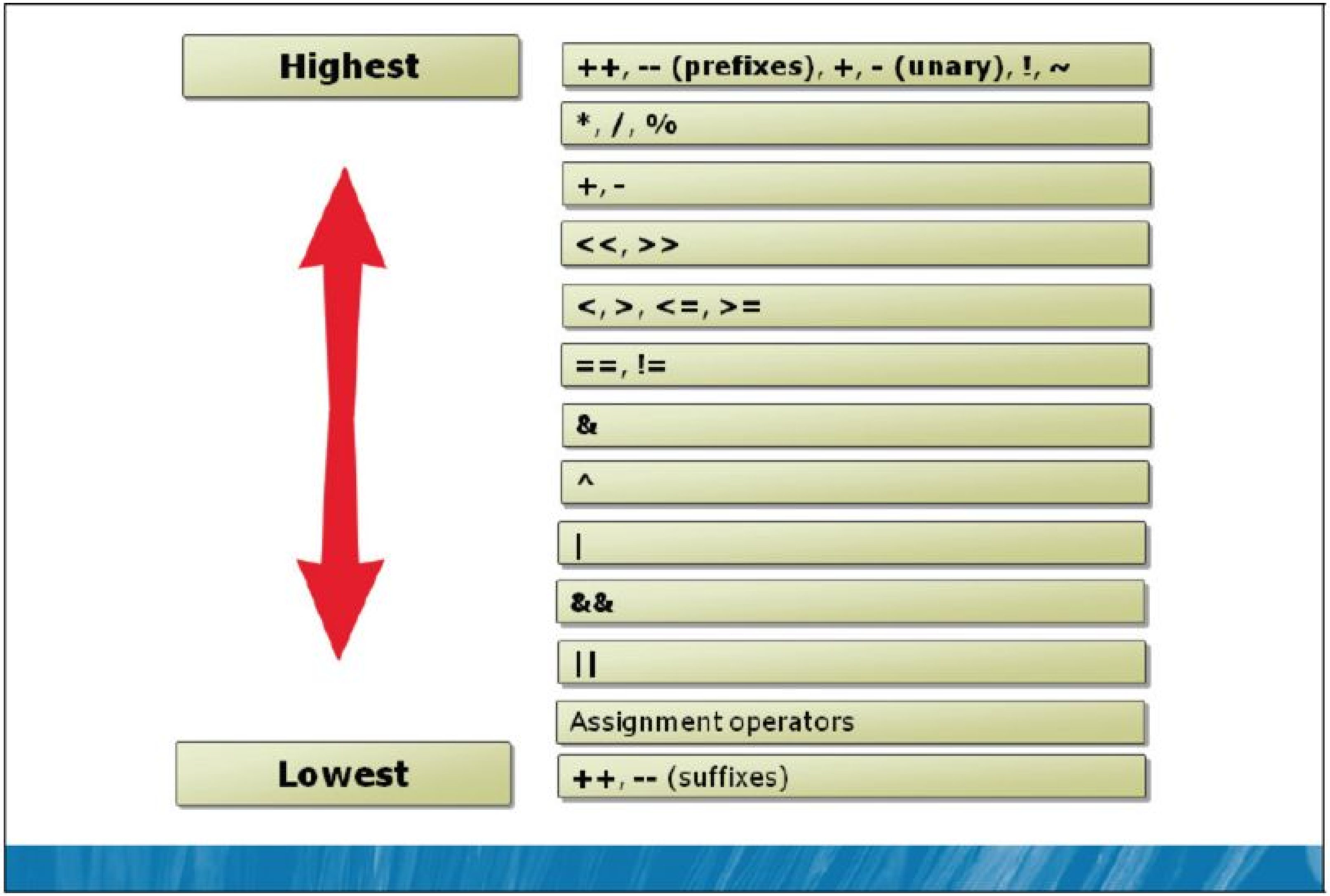
```
using System;
...
Console.WriteLine("Hello there!“);
```

C# Operators



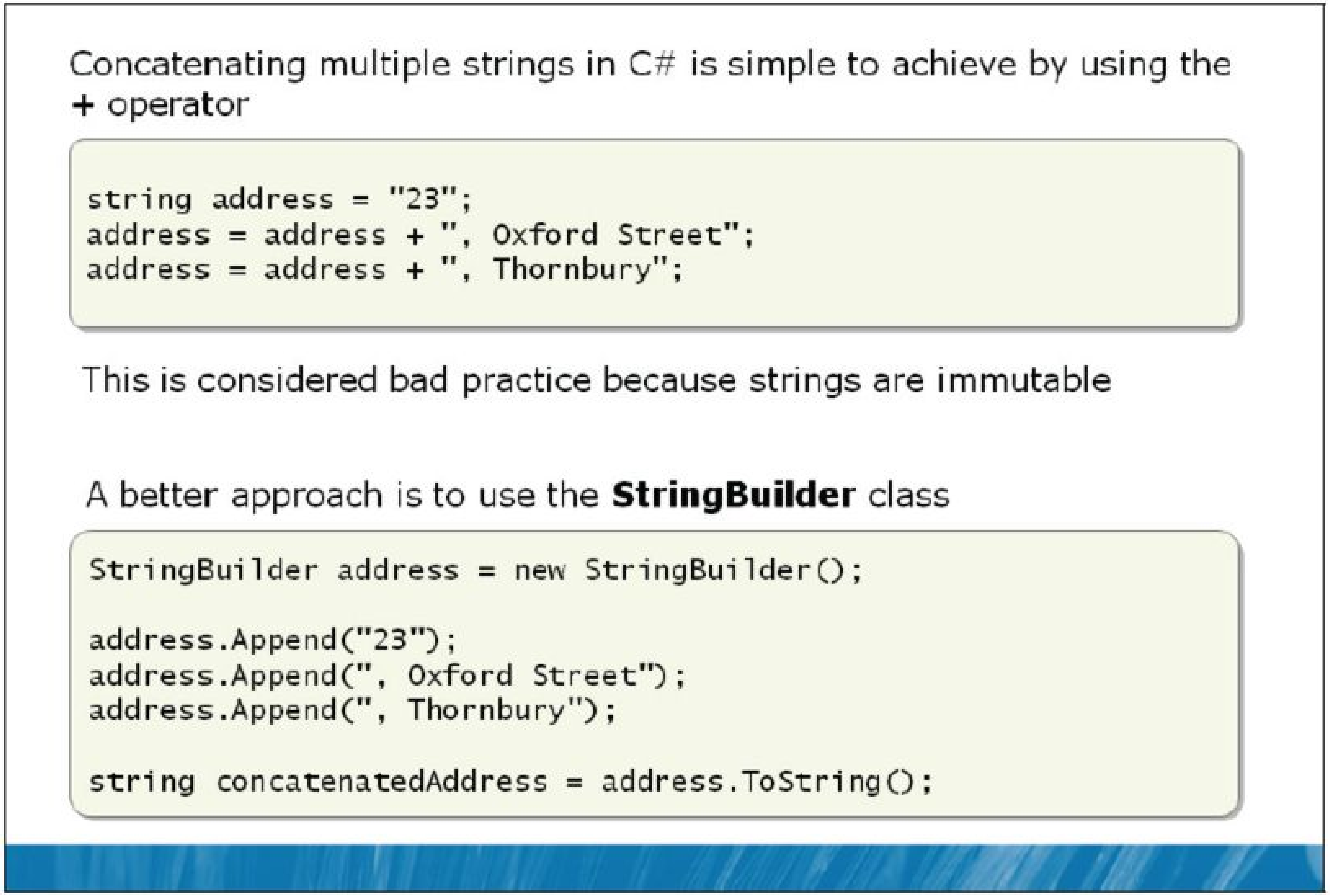
Operator type	Operators
Arithmetic	+, -, *, /, %
Increment, decrement	++, --
Comparison	==, !=, <, >, <=, >=, is
String concatenation	+
Logical/bitwise operations	&, |, ^, !, ~, &&, ||
Indexing (counting starts from element 0)	[ ]
Casting	( ), as
Assignment	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, ??

Specifying Operator Precedence



++, -- (prefixes), +, - (unary), !, ~
*, /, %
+, -
<<, >>
<, >, <=, >=
==, !=
&
^
|
&&
||
Assignment operators
++, -- (suffixes)

Best Practices for Performing String Concatenation



Concatenating multiple strings in C# is simple to achieve by using the + operator

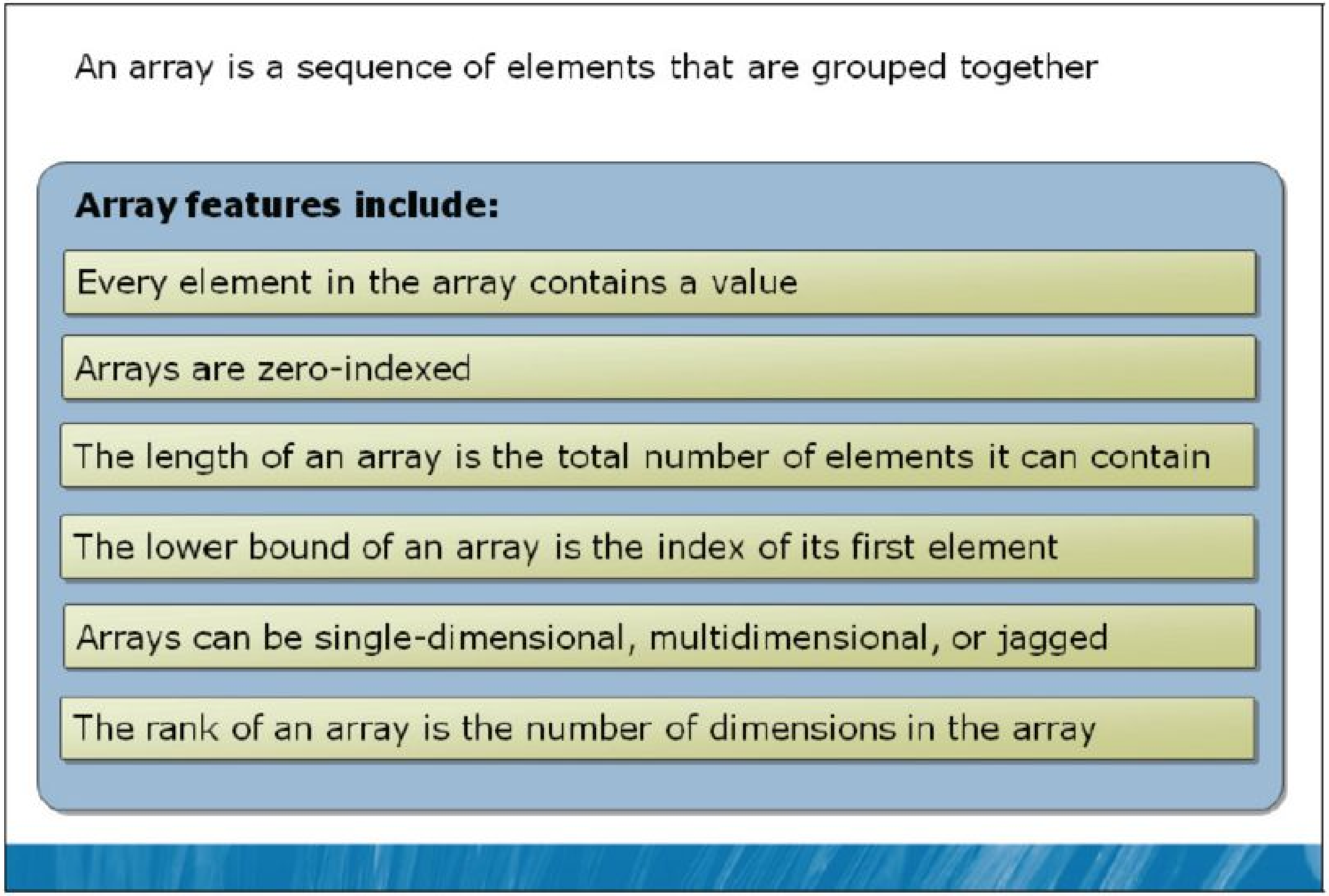
```
string address = "23";
address = address + ", Oxford Street";
address = address + ", Thornbury";
```

This is considered bad practice because strings are immutable

A better approach is to use the **StringBuilder** class

```
StringBuilder address = new StringBuilder();

address.Append("23");
address.Append(", Oxford Street");
address.Append(", Thornbury");

string concatenatedAddress = address.ToString();
```

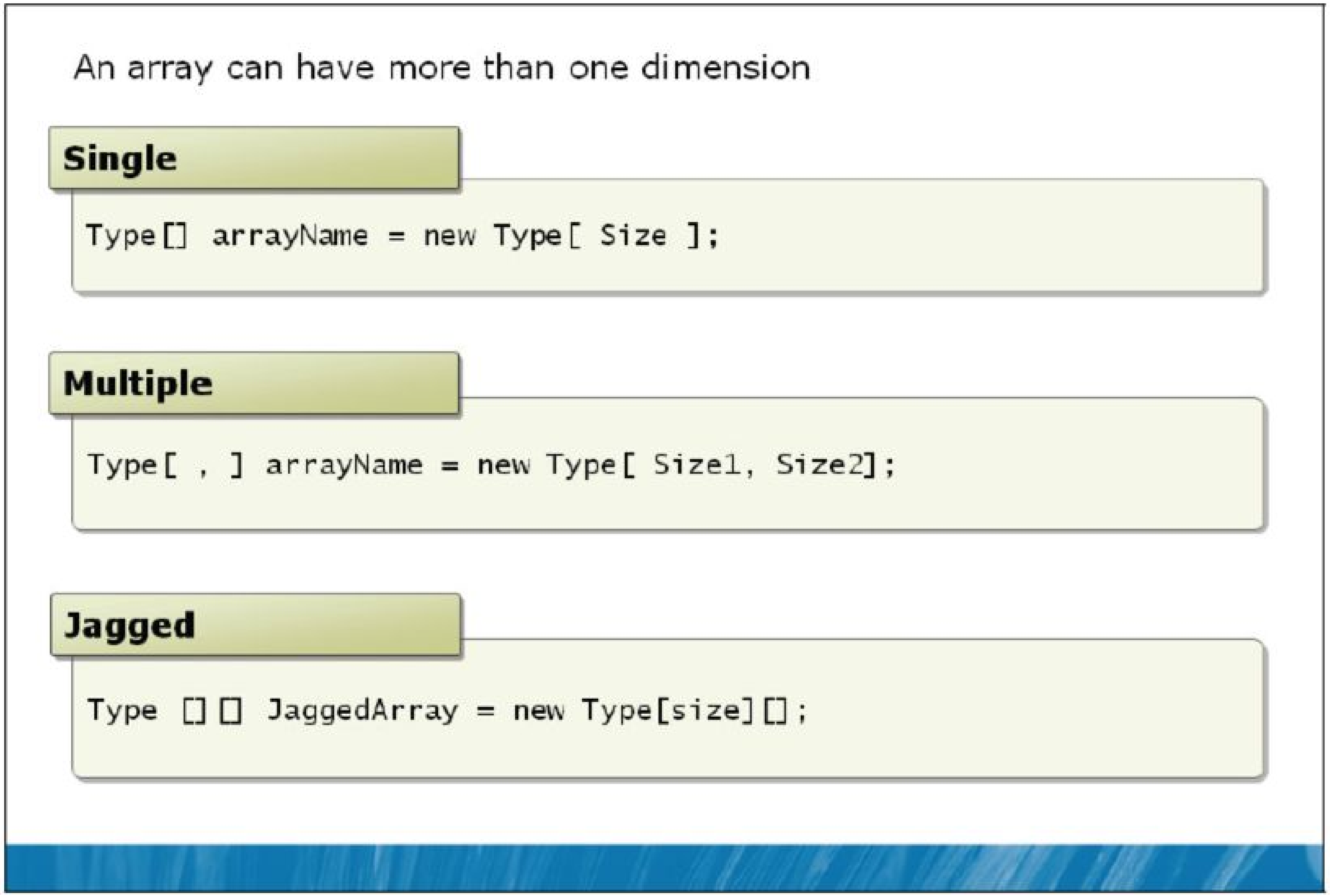
What Is an Array?


An array is a sequence of elements that are grouped together

Array features include:

- Every element in the array contains a value
- Arrays are zero-indexed
- The length of an array is the total number of elements it can contain
- The lower bound of an array is the index of its first element
- Arrays can be single-dimensional, multidimensional, or jagged
- The rank of an array is the number of dimensions in the array

Creating and Initializing Arrays



An array can have more than one dimension

Single

```
Type[] arrayName = new Type[ Size ];
```

Multiple

```
Type[ , ] arrayName = new Type[ Size1, Size2];
```

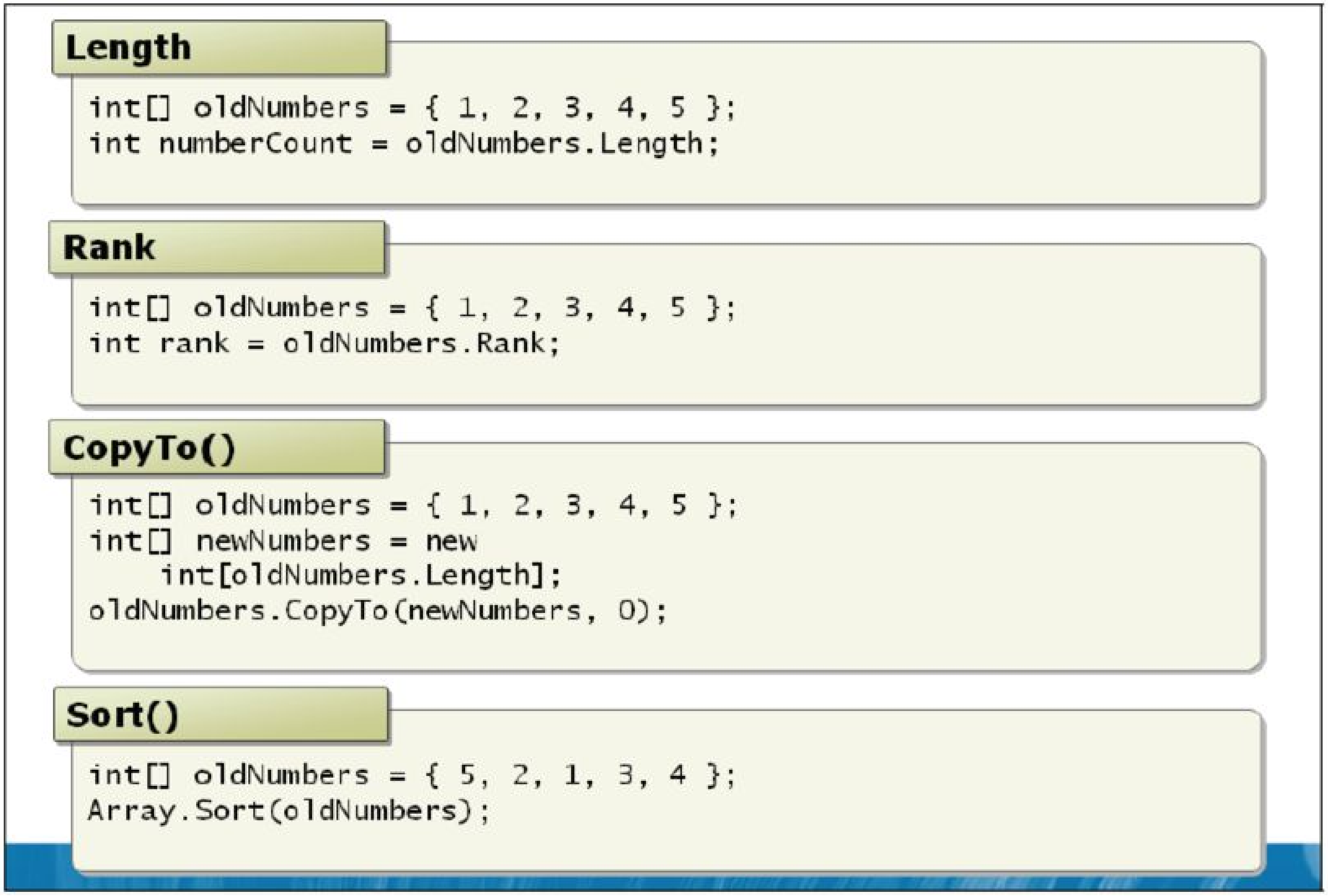
Jagged

```
Type [] [] JaggedArray = new Type[size][];
```

Implicitly Typed Arrays

- var numbers = new[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

Common Properties and Methods Exposed by Arrays



Length

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int numberCount = oldNumbers.Length;
```

Rank

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int rank = oldNumbers.Rank;
```

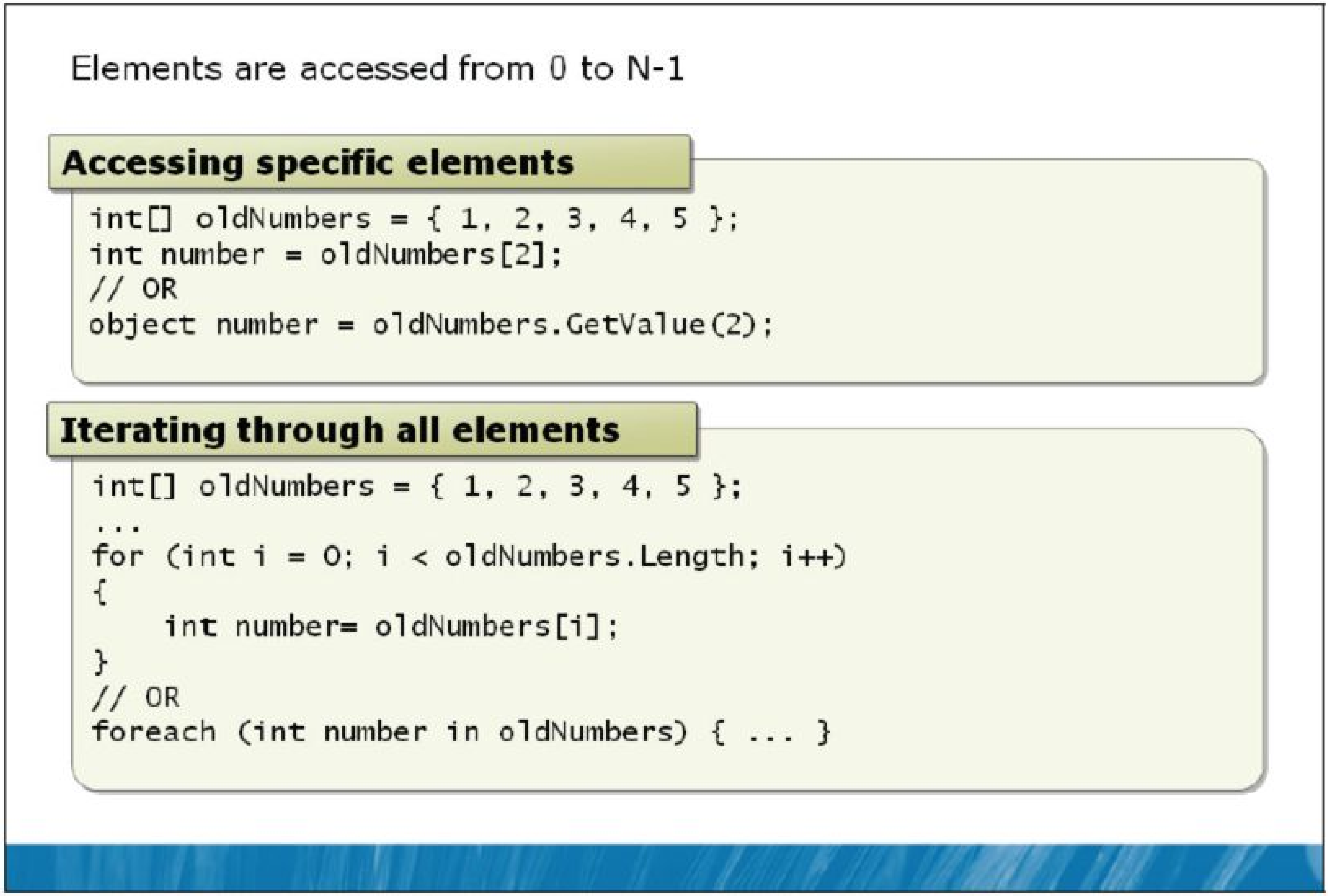
CopyTo()

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int[] newNumbers = new
    int[oldNumbers.Length];
oldNumbers.CopyTo(newNumbers, 0);
```

Sort()

```
int[] oldNumbers = { 5, 2, 1, 3, 4 };
Array.Sort(oldNumbers);
```

Accessing Data in an Array



Elements are accessed from 0 to N-1

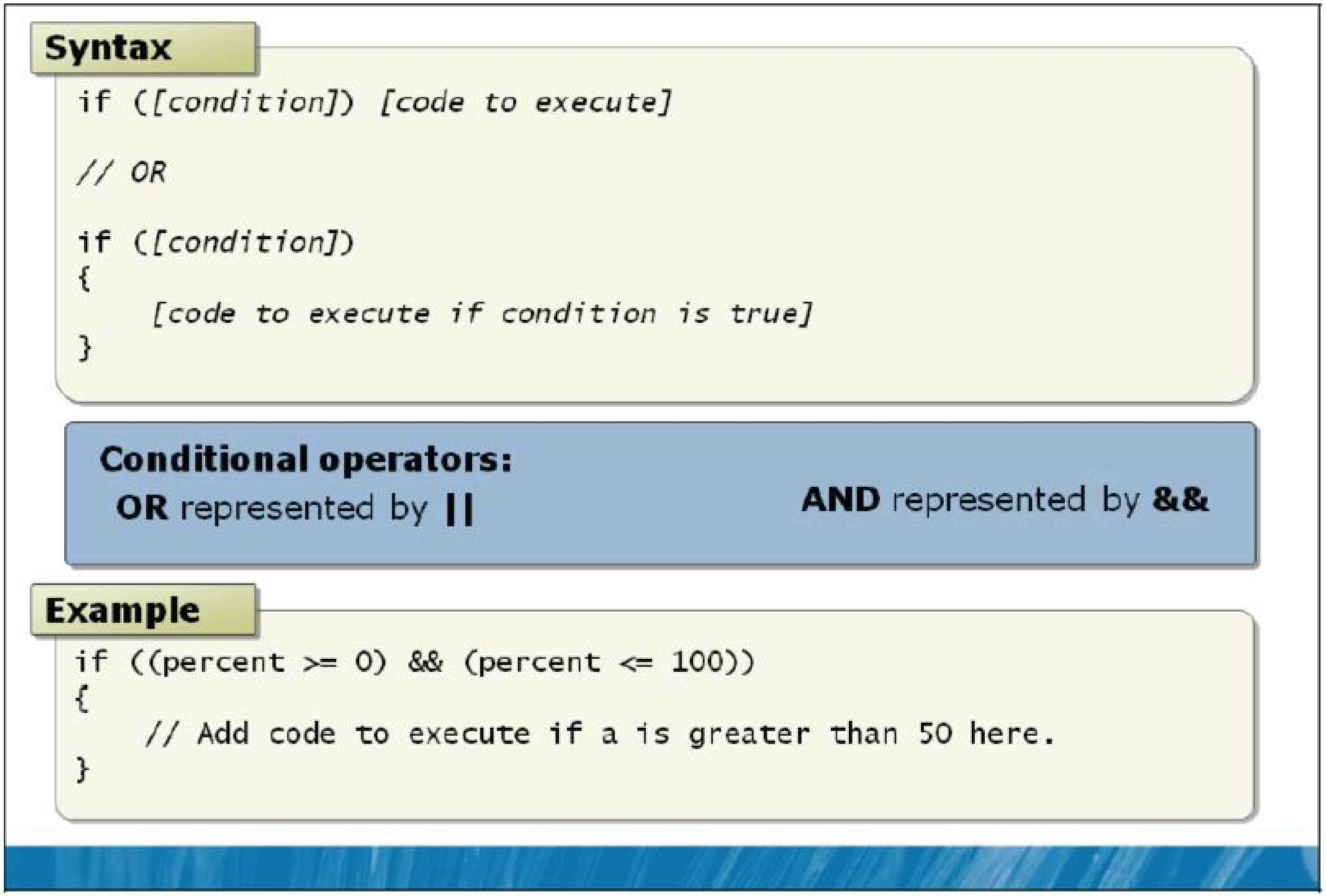
Accessing specific elements

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
int number = oldNumbers[2];
// OR
object number = oldNumbers.GetValue(2);
```

Iterating through all elements

```
int[] oldNumbers = { 1, 2, 3, 4, 5 };
...
for (int i = 0; i < oldNumbers.Length; i++)
{
     int number= oldNumbers[i];
}
// OR
foreach (int number in oldNumbers) { ... }
```

Using Decision Statements



Syntax

```
if ([condition]) [code to execute]

// OR

if ([condition])
{
     [code to execute if condition is true]
}
```

Conditional operators:

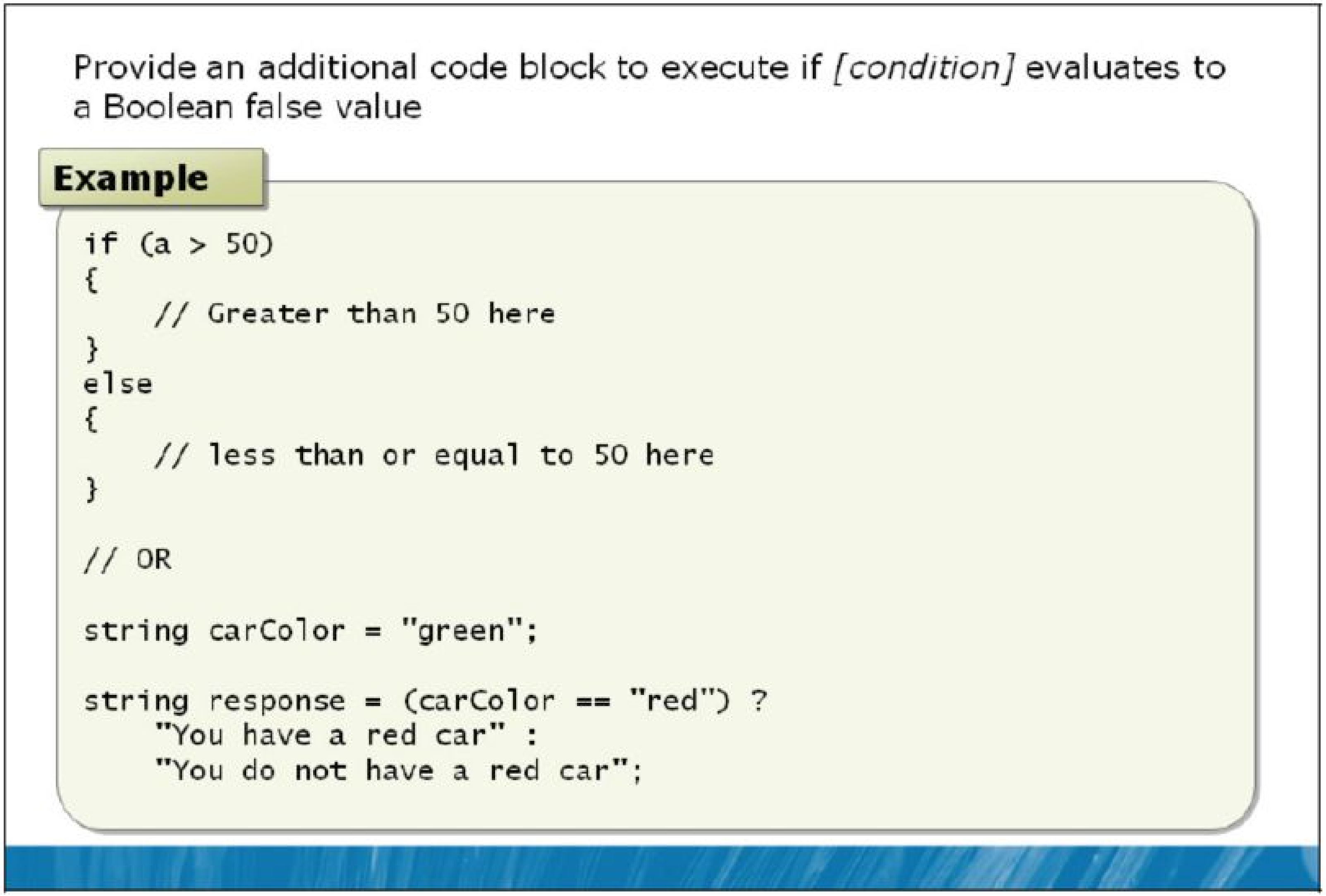
OR represented by **||**

AND represented by **&&**

Example

```
if ((percent >= 0) && (percent <= 100))
{
    // Add code to execute if a is greater than 50 here.
}
```

Using Either-Or If Statements



Provide an additional code block to execute if *[condition]* evaluates to a Boolean false value

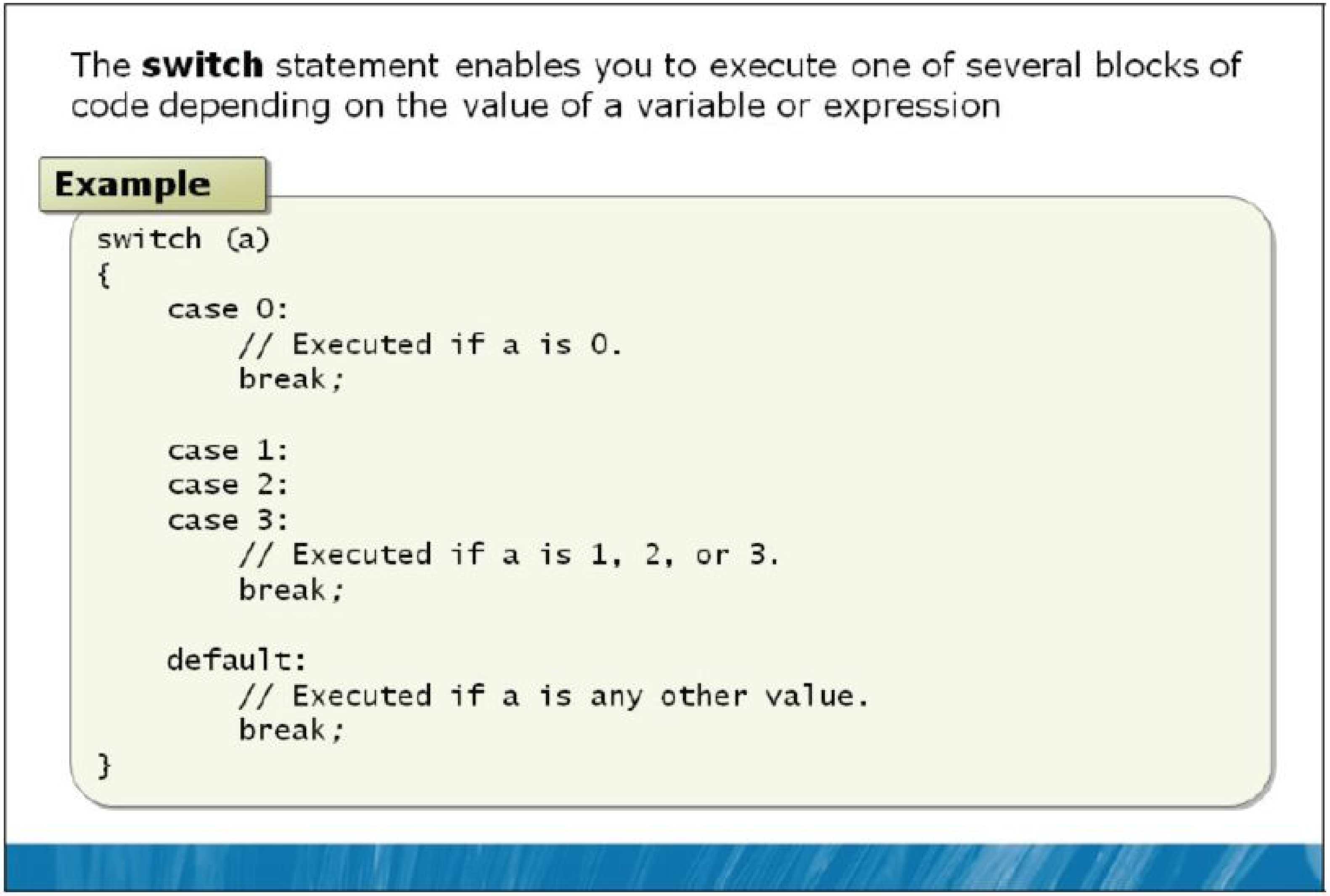
Example

```
if (a > 50)
{
    // Greater than 50 here
}
else
{
    // less than or equal to 50 here
}

// OR

string carColor = "green";

string response = (carColor == "red") ?
    "You have a red car" :
    "You do not have a red car";
```

Using the ?: Operator

- Type result = [condition] ? [true expression] : [false expression]
- string carColor = "green”;
- string response = (carColor == "red") ?"You have a red car" :"You do not have a red car";

Using the Switch Statement

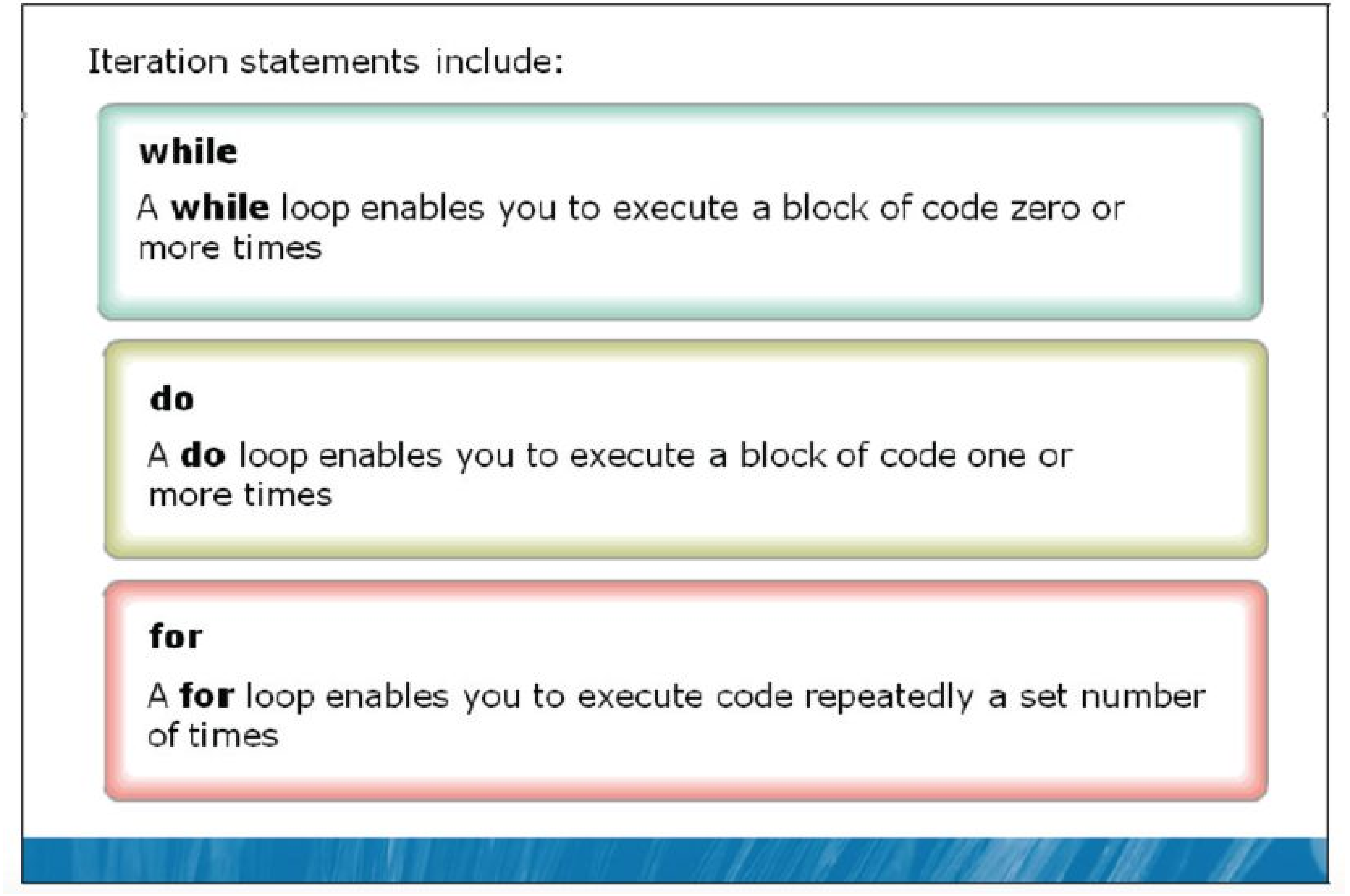


The **switch** statement enables you to execute one of several blocks of code depending on the value of a variable or expression

Example

```
switch (a)
{
    case 0:
        // Executed if a is 0.
        break;

    case 1:
    case 2:
    case 3:
        // Executed if a is 1, 2, or 3.
        break;

    default:
        // Executed if a is any other value.
        break;
}
```

Using Iteration Statements



Iteration statements include:

while

A **while** loop enables you to execute a block of code zero or more times

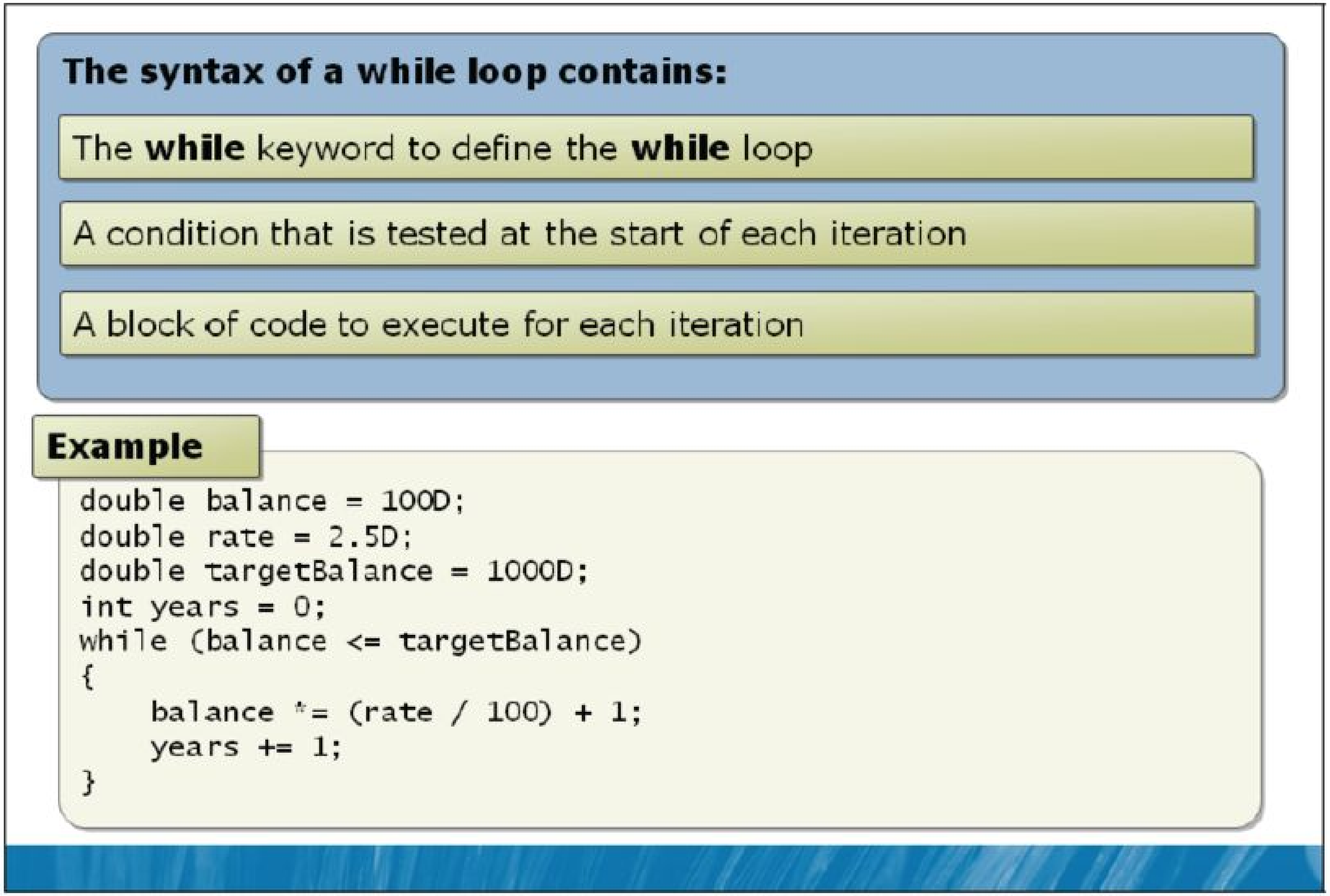
do

A **do** loop enables you to execute a block of code one or more times

for

A **for** loop enables you to execute code repeatedly a set number of times

Using the While Statement



The syntax of a while loop contains:

The **while** keyword to define the **while** loop

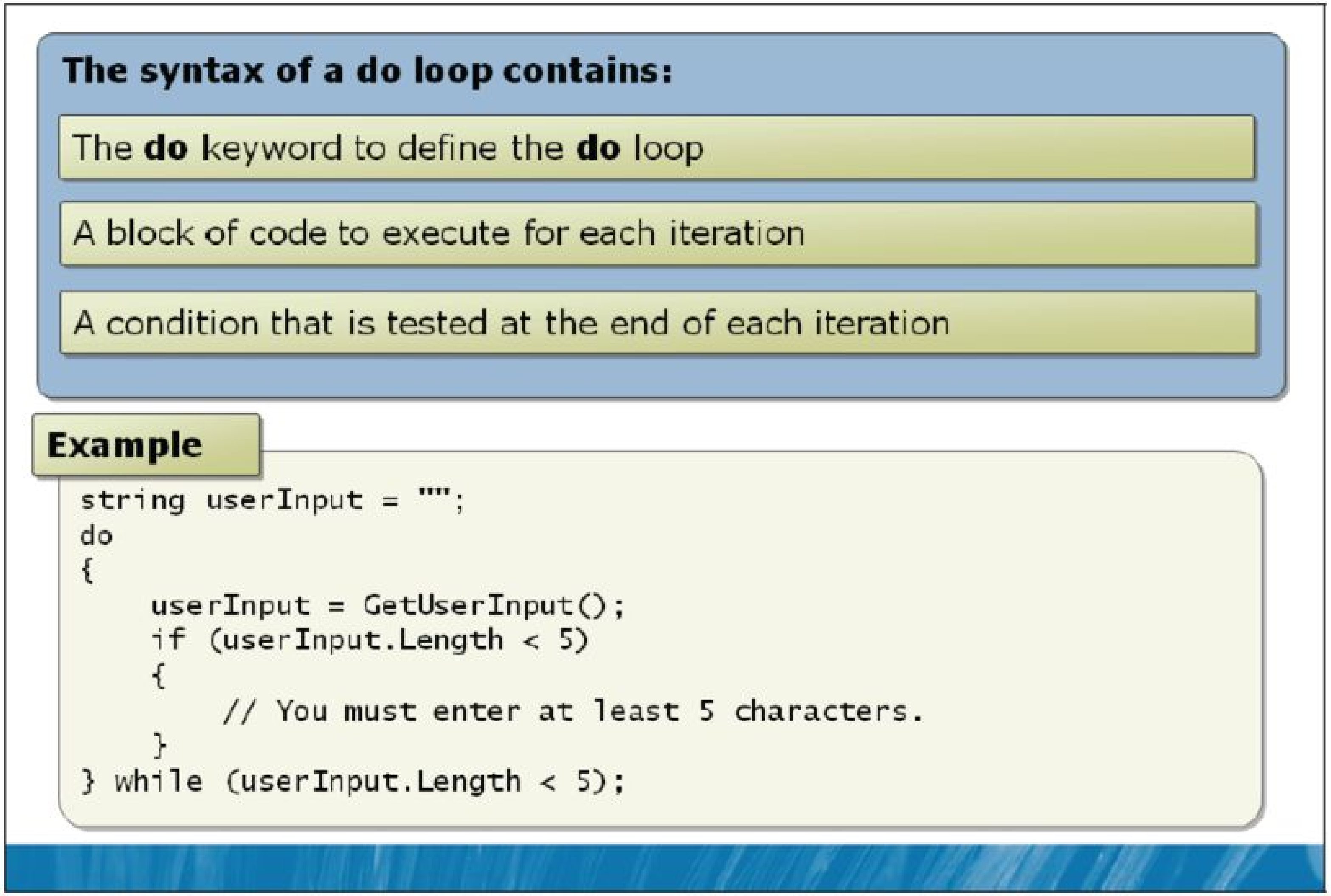
A condition that is tested at the start of each iteration

A block of code to execute for each iteration

Example

```
double balance = 100D;
double rate = 2.5D;
double targetBalance = 1000D;
int years = 0;
while (balance <= targetBalance)
{
    balance *= (rate / 100) + 1;
    years += 1;
}
```

Using the Do Statement



The syntax of a do loop contains:

The **do** keyword to define the **do** loop

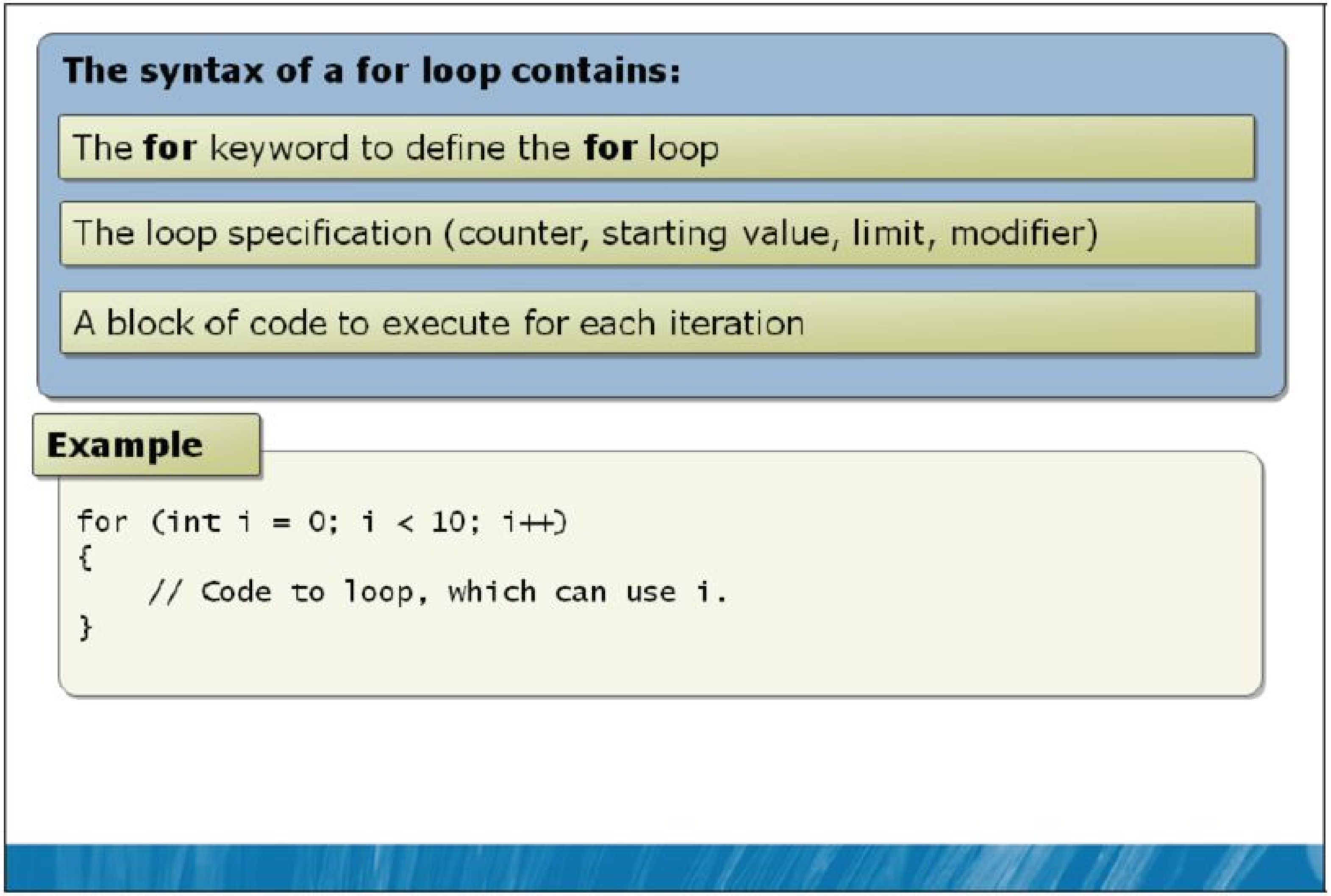
A block of code to execute for each iteration

A condition that is tested at the end of each iteration

Example

```
string userInput = "";
do
{
    userInput = GetUserInput();
    if (userInput.Length < 5)
    {
        // You must enter at least 5 characters.
    }
} while (userInput.Length < 5);
```

Using the For Statement



The syntax of a for loop contains:

The **for** keyword to define the **for** loop

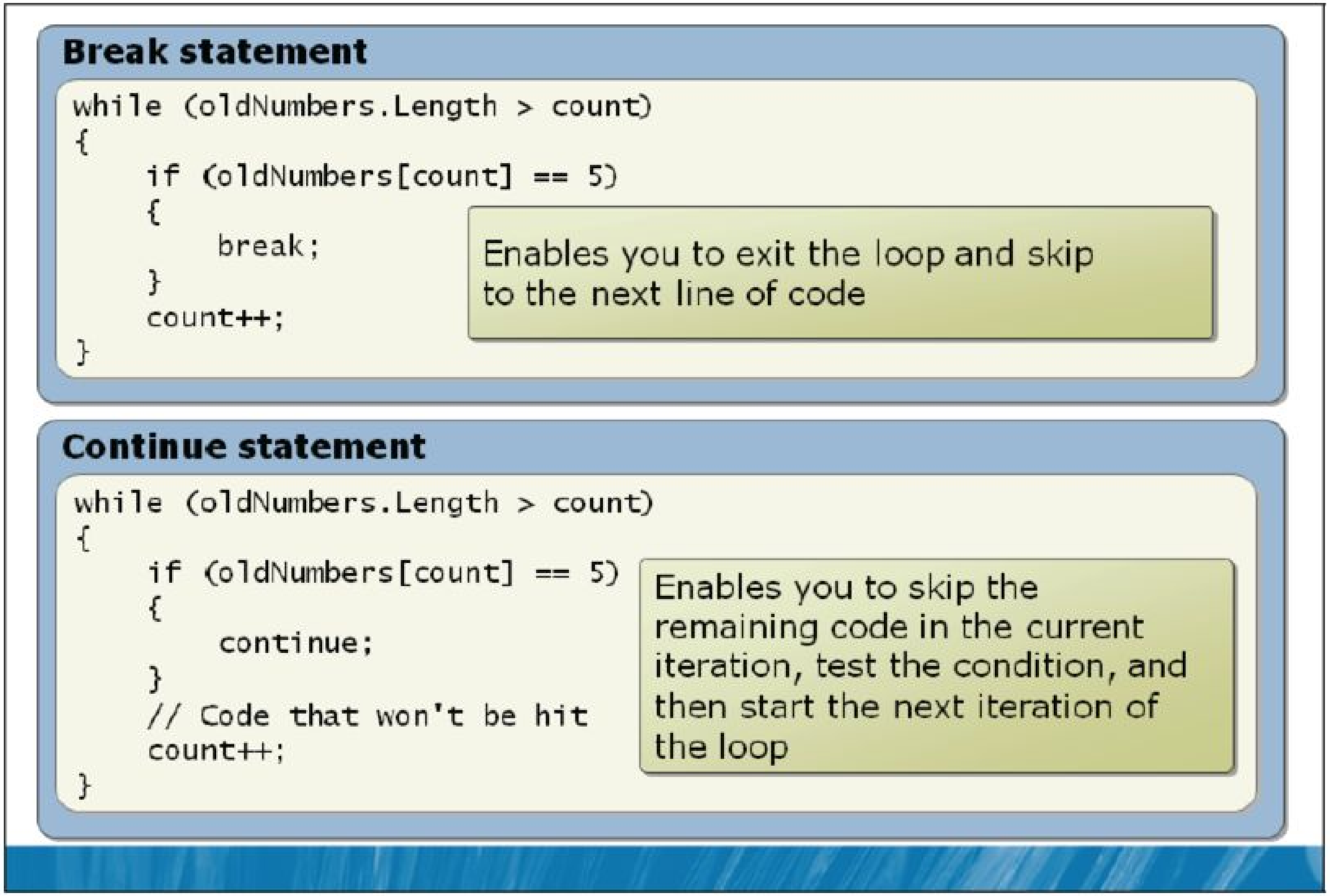
The loop specification (counter, starting value, limit, modifier)

A block of code to execute for each iteration

Example

```
for (int i = 0; i < 10; i++)
{
    // Code to loop, which can use i.
}
```

Break and Continue Statements



Break statement

```
while (oldNumbers.Length > count)
{
    if (oldNumbers[count] == 5)
    {
        break;
    }
    count++;
}
```

Enables you to exit the loop and skip to the next line of code

Continue statement

```
while (oldNumbers.Length > count)
{
    if (oldNumbers[count] == 5)
    {
        continue;
    }
    // Code that won't be hit
    count++;
}
```

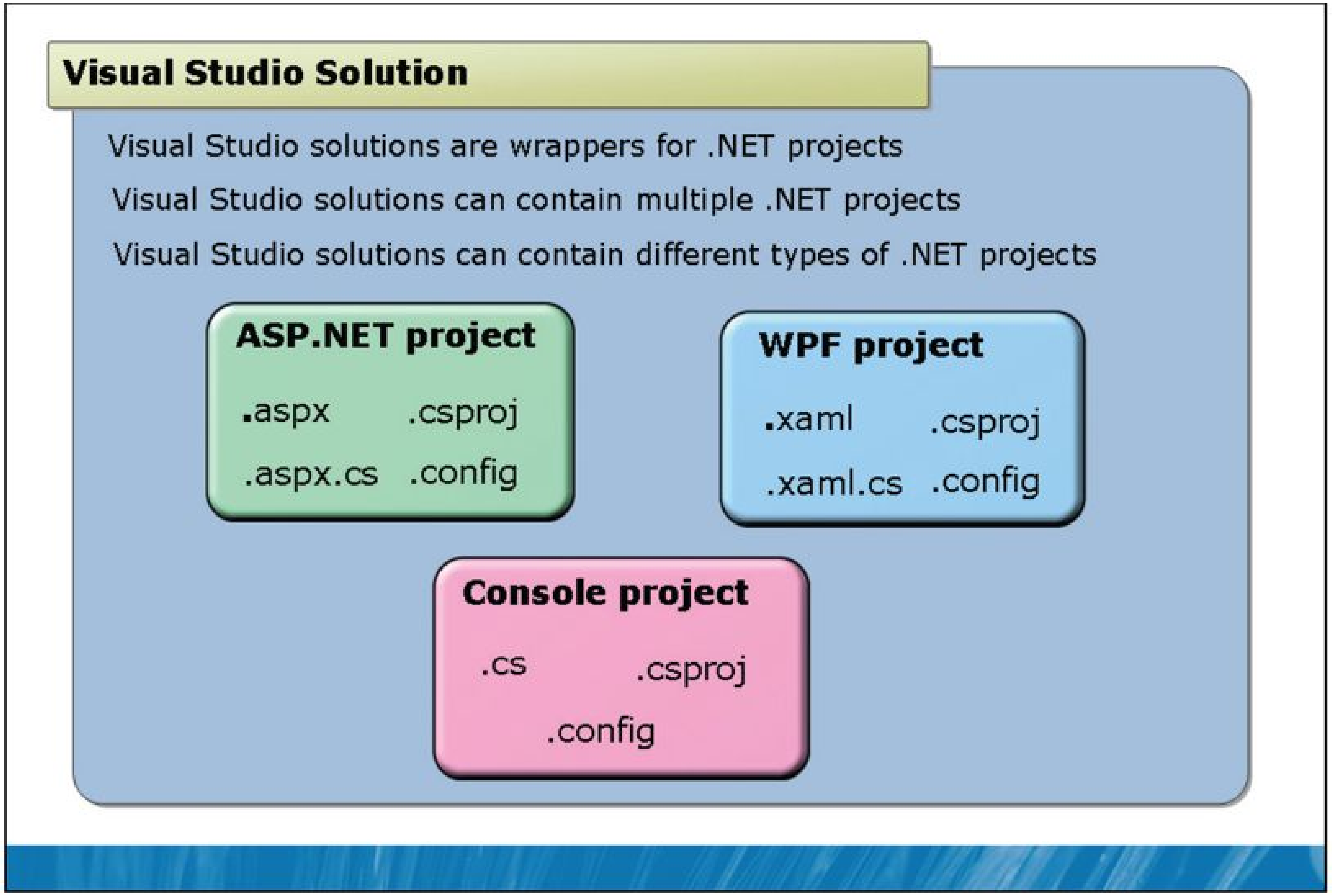
Enables you to skip the remaining code in the current iteration, test the condition, and then start the next iteration of the loop

- Do.. While
- Foreach

Exercise Questions

- Write a program that print odd numbers from 1 to 100;
- Write a program that accept 5 integer numbers and printout the largest number among them.
- Write a program that sort the following arrays in ascending order
  - [8,7,2,3,1]
- The iteration expression in a **for** loop need not always alter the loop control variable by a fixed amount. Instead, the loop control variable can change in any arbitrary way. Using this concept, write a program that uses a **for** loop to generate and display the progression 1, 2, 4, 8, 16, 32, and so on. write only the for loop.

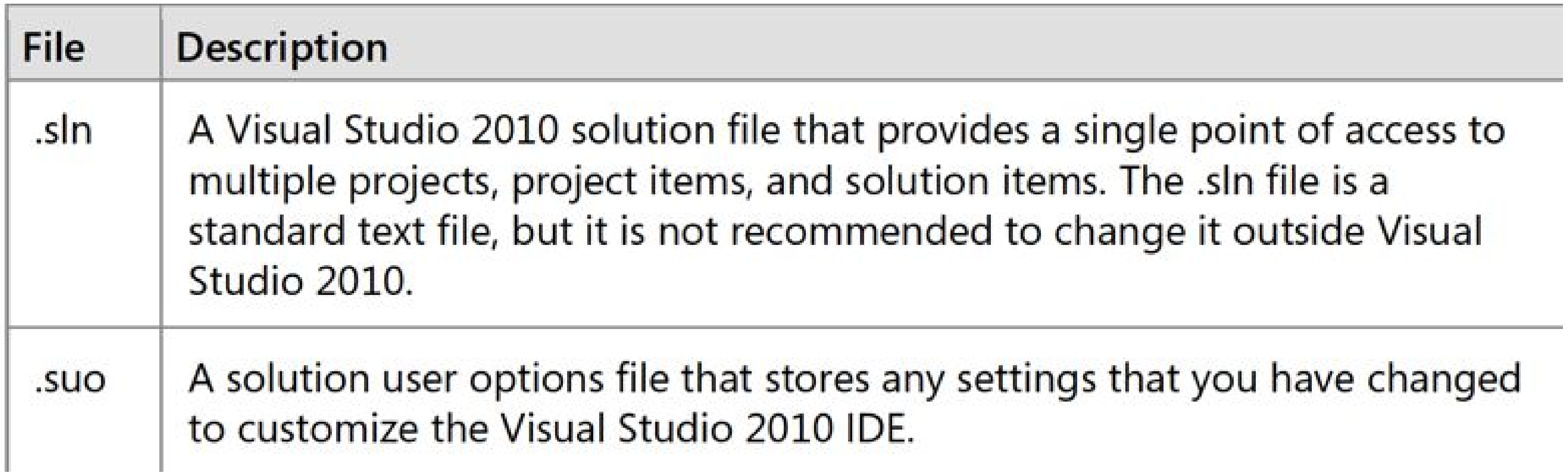
The Structure of Visual Studio Projects and Solutions


Visual Studio Solution
Visual Studio solutions are wrappers for .NET projects
Visual Studio solutions can contain multiple .NET projects
Visual Studio solutions can contain different types of .NET projects
ASP.NET project
.aspx
.csproj
.aspx.cs
.config
WPF project
.xaml
.csproj
.xaml.cs
.config
Console project
.cs
.csproj
.config

Cont..

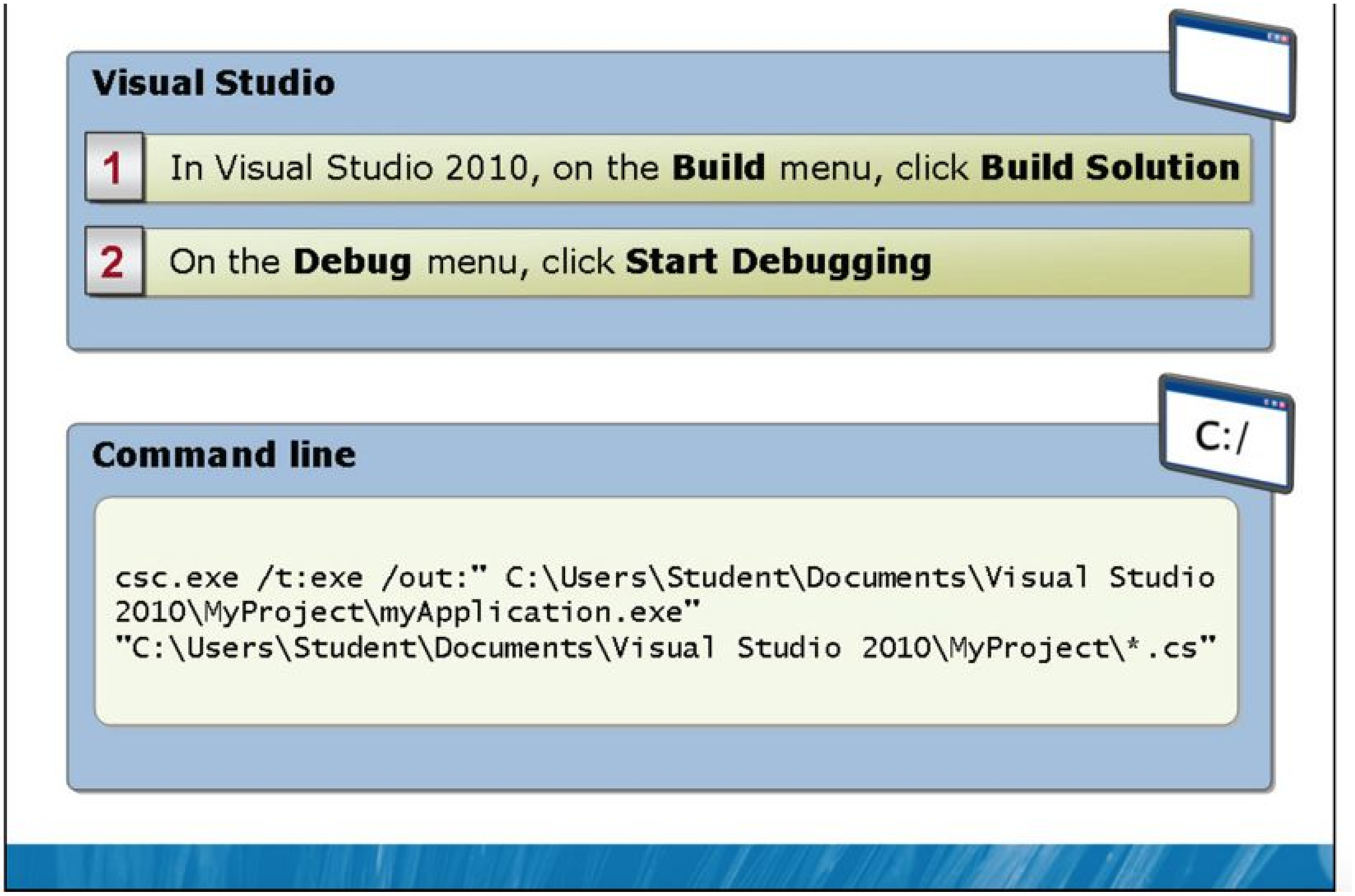
File	Description
.cs	Code files that can belong to a single project solution. This type of file can represent any of the following: • Modules • Windows Forms files • Class files
.csproj	Project files that can belong to multiple project solutions. The .csproj file also stores settings for the project, such as the output path for the build output and the target platform.
.aspx	Files that represent ASP.NET Web pages. An ASP.NET file can contain your Visual C# code or you can use an accompanying .aspx.cs file to store your code in addition to the page markup.
.config	Configuration files are XML-based files that you can use to store application-level settings such as database connection strings, which you can then modify without recompiling your application.
.xaml	XAML files are used in WPF and Microsoft Silverlight® applications to define user interface elements.

Visual Studio Solutions



File	Description
.sln	A Visual Studio 2010 solution file that provides a single point of access to multiple projects, project items, and solution items. The .sln file is a standard text file, but it is not recommended to change it outside Visual Studio 2010.
.suo	A solution user options file that stores any settings that you have changed to customize the Visual Studio 2010 IDE.

Building and Running a .NET Framework Application



Visual Studio

1 In Visual Studio 2010, on the **Build** menu, click **Build Solution**

2 On the **Debug** menu, click **Start Debugging**

Command line

```
csc.exe /t:exe /out:" C:\Users\Student\Documents\Visual Studio
2010\MyProject\myApplication.exe"
"C:\Users\Student\Documents\Visual Studio 2010\MyProject\*.cs"
```

if the input array is	**return**
{1, 2, 3, 0}	1 (because 1+2 == 3+0 == 3)
{1, 2, 2, 1, 3, 0}	1 (because 1+2 == 2+1 == 3+0 == 3)
{1, 1, 2, 2}	0 (because 1+1 == 2 != 2+2)
{1, 2, 1}	0 (because array does not have an even number of elements)
{}	1

- Gashaw.tsegaye@aau.edu.et