Chapter 2: Core Event-Driven Programming

Ref. Murach’s C# (7th Ed.), Ch. 4–7; Deitel & Deitel (2017), Ch. 4–6, 12; csharpkey.com (Events and Delegates)

Chapter 2: Core Event-Driven Programming

- Review of C# basics (variables, data types, control structures)
- Delegates and event handlers
- Event-driven design patterns
- Working with arrays, lists, and strings
- Applying OOP principles (classes, inheritance) in GUI apps

Review of C# basics (variables, data types, control structures)

- Review core C# elements: variables, data types, and control structures.
- These form the building blocks for handling data and logic in your programs.
- How to Work with Numeric and String Data
- Start your mastery of the C# language
- Learn arithmetic operations on numeric data, working with string data, and converting data types

Overview of Data Types in C#

- C# provides a wide range of built-in data types categorized mainly as:
  - Value Types – store data directly (e.g., int, double, bool, char).
  - Reference Types – store references to data (e.g., string, arrays, objects).
- Data types define size, range, and operations allowed.
- Defined under System namespace (e.g., System.Int32).
- **Variables and Data Types**
  - A variable stores a value that can change as the program executes.
  - You must declare a variable's type and name, and you can optionally initialize it with a value.

Numeric Data Types

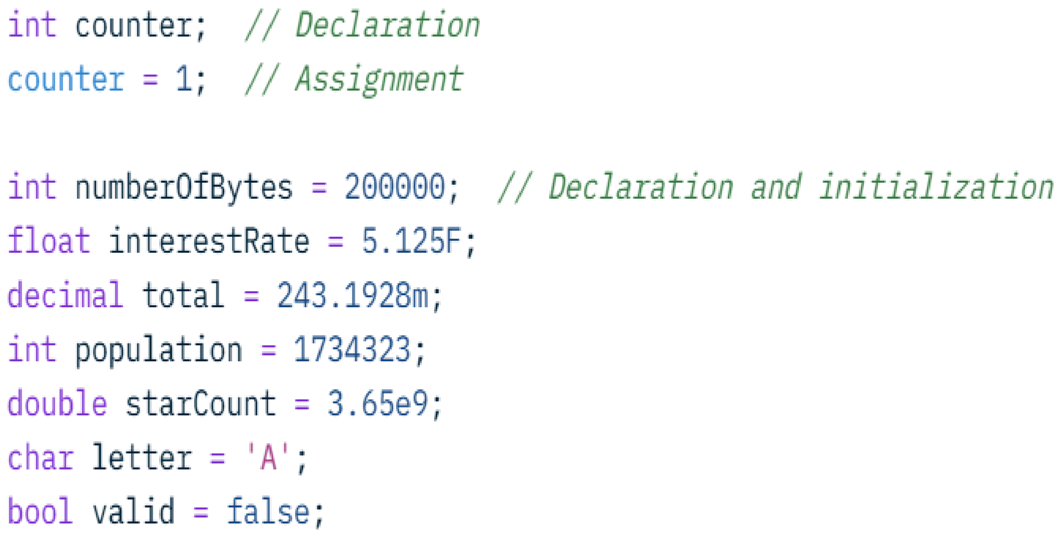
- Integral types: byte, short, int, long – store whole numbers.
- Floating-point types: float, double, decimal – store real numbers.
- Numeric literals can include suffixes: 5.0f (float), 5.0d (double).
- Supports arithmetic operators: +, -, *, /, %, ++, --.
- Overflow and rounding errors may occur; use checked blocks to detect overflow.

Declaring and Initializing Variables:

- A variable stores a value that can change during execution
- Must declare type and name, then initialize with a value
- Two ways:

1. Separate statements: Declare then assign
   - Use two statements: type variableName; followed by variableName = value;.
2. Single statement: Declare and assign
   - Or in one statement: type variableName = value;.

- Always initialize before use to avoid build errors
- Use var for inferred types (useful with tuples/LINQ)
- Naming: camel notation (lowercase first word, uppercase subsequent)
- Literals: Direct values assigned to variables



```
int counter;  // Declaration
counter = 1;  // Assignment

int numberOfBytes = 200000;  // Declaration and initialization
float interestRate = 5.125F;
decimal total = 243.1928m;
int population = 1734323;
double starCount = 3.65e9;
char letter = 'A';
bool valid = false;
```

- Decimal literals default to double; use m/M for decimal, f/F for float
- Underscores (_) as digit separators for readability (C# 7.0+):
  - int hex = 0xFF_FF; // 65535
  - int binary = 0b1010_0110; // 166
  - int population = 1_275_000_000;
  - double distance = 9_460_730_472_580.8; // light-years in meters
- Scientific notation: e/E for powers of 10 (e.g., 3.65e9 = 3.65 x 10^9)
- char literals in single quotes ('A')
- bool: true or false keywords
- Multiple variables in one statement: Separate with commas

Notes on Variables:

- Variables change as program executes
- Declare type and assign initial value
- Initial values: 0 for ints, 0.0 for decimals, false for bool
- Use commas for multiple declarations/assignments
- Data type keywords: All lowercase
- Naming: camel notation
- Meaningful, easy-to-remember names

- Built-in Value Types: C# uses .NET's Common Type System (CTS) for data types, which are aliases for .NET types.

C# Keyword	Bytes	.NET Type	Description
byte	1	Byte	A positive integer value from 0 to 255
sbyte	1	SByte	A signed integer value from -128 to 127
short	2	Int16	An unsigned integer from -32,768 to 32,767
ushort	2	UInt16	An unsigned integer from 0 to 65,535
int	4	Int32	An integer from -2,147,483,648 to 2,147,483,647
uint	4	UInt32	An unsigned integer from 0 to 4,294,967,295
long	8	Int64	An integer from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	8	UInt64	An unsigned integer from 0 to 18,446,744,073,709,551,615
float	4	Single	A non-integer number with approximately 7 significant digits
double	8	Double	A non-integer number with approximately 14 significant digits
decimal	16	Decimal	A non-integer number with up to 28 significant digits (integer and fraction); can represent values up to 7.9228 x 10^28
char	2	Char	A single Unicode character
bool	1	Boolean	A true or false value

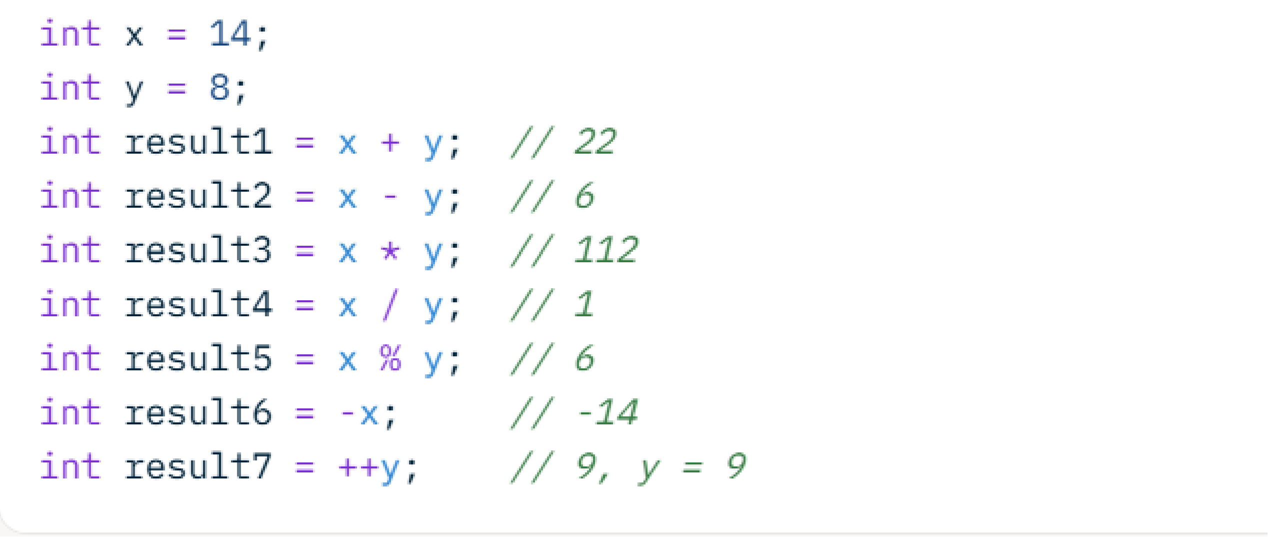
- Summaries of the value types that .NET provides
  - Use keywords to refer to data types
  - First 11 types for numbers, last 2 for characters and true/false values
  - Integers: Whole numbers without decimal places
  - Use int most often
  - Use long for large values
  - Use short or byte for small values to save resources
  - Unsigned versions for positive numbers only
- **Constants**: Constants store unchanging values.
- Declare with const keyword: const type ConstantName = value;.
  - Example:
    - const int DaysInNovember = 30;
    - const decimal SalesTax = .075m;
- Naming: Use Pascal notation (capitalize first letter of each word) for constant names.

Arithmetic Expressions and Operators:

- Use operators or arithmetic operators to perform calculations on numeric data.
- Examples include addition, subtraction, multiplication, division, and more.
- Shortcut assignment operators like +=, -= for concise code.

Operator	Name	Description
+	Addition	Adds two operands.
-	Subtraction	Subtracts the right operand from the left operand.
*	Multiplication	Multiplies the right operand from the left operand.
/	Division	Divides the right operand into the left operand. If both operands are integers, the result is an integer.
%	Modulus	Returns the value that is left over after dividing the right operand into the left operand.
+	Positive sign	Returns the value of the operand.
-	Negative sign	Changes a positive value to negative, and vice versa.
++	Increment	Adds 1 to the operand (x = x + 1).
--	Decrement	Subtracts 1 from the operand (x = x - 1).

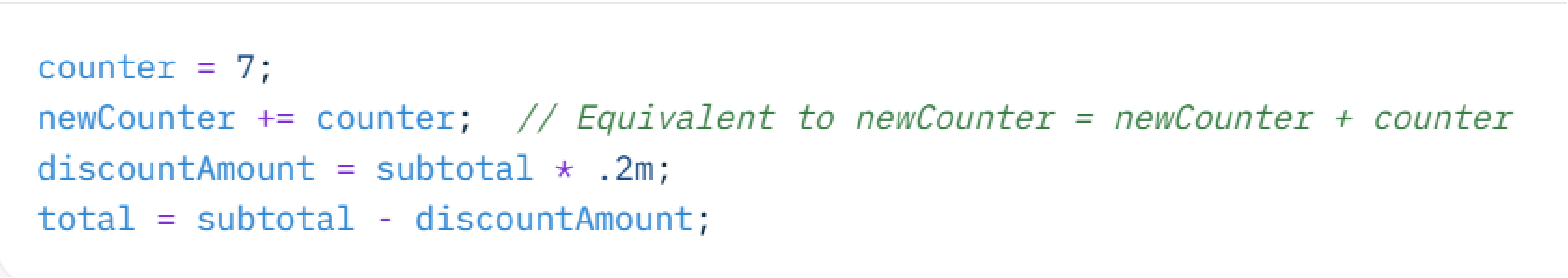
- Examples of use of operators in C#



```
int x = 14;
int y = 8;
int result1 = x + y;  // 22
int result2 = x - y;  // 6
int result3 = x * y;  // 112
int result4 = x / y;  // 1
int result5 = x % y;  // 6
int result6 = -x;     // -14
int result7 = ++y;    // 9, y = 9
```

- Combine variables, literals, and operators.
- Use shortcut operators for efficiency, e.g., counter += 1 is same as counter = counter + 1.
- Division with integers truncates decimal part.

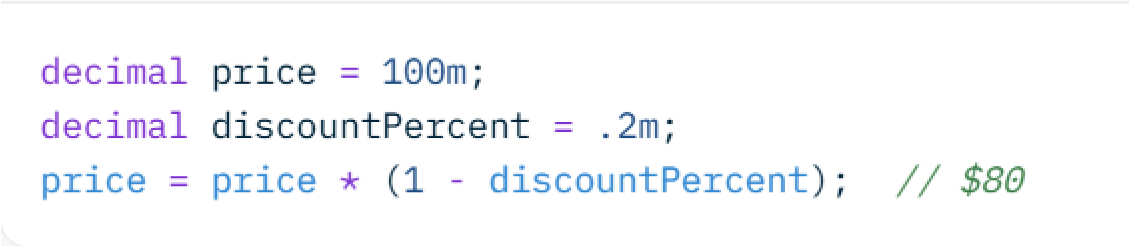
- **Assignment Statements:**
  - Assign values using =.
  - Shortcut operators: +=, -=, *=, /=, %=.



```
counter = 7;
newCounter += counter;  // Equivalent to newCounter = newCounter + counter
discountAmount = subtotal * .2m;
total = subtotal - discountAmount;
```

Order of Precedence:

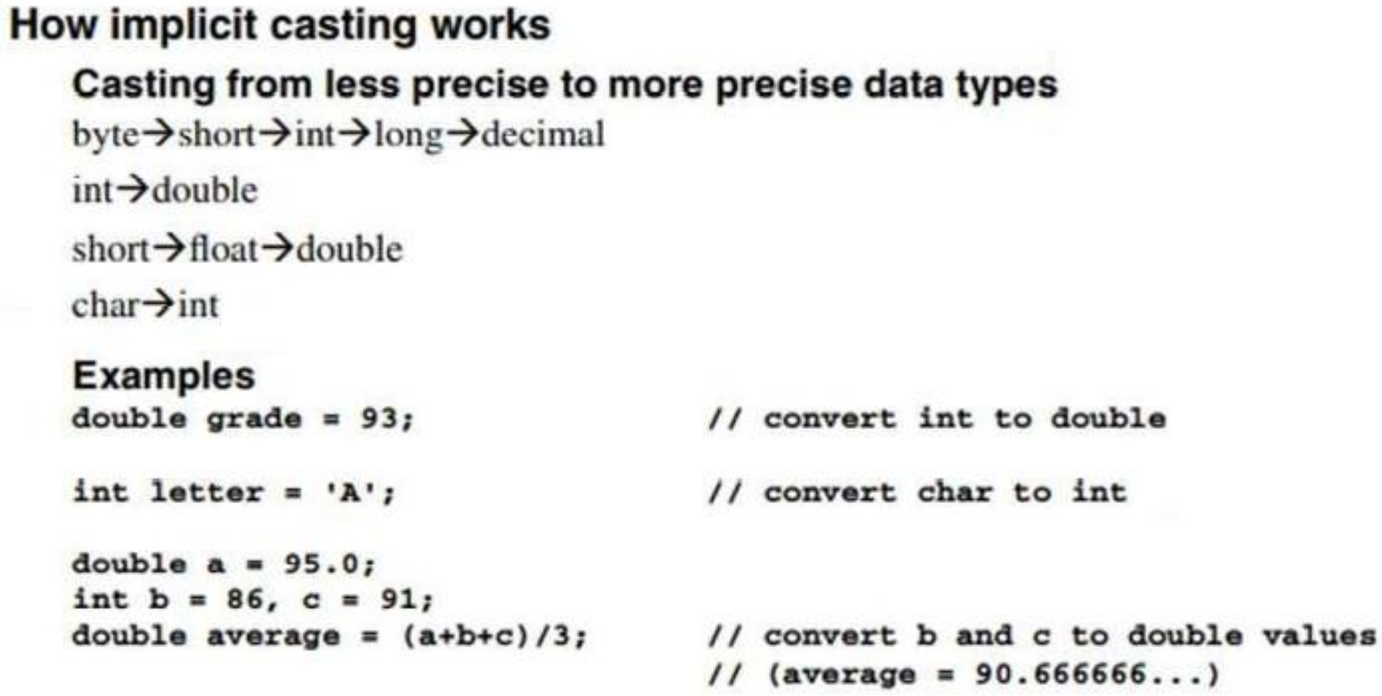
- Operators have precedence levels that determine evaluation order.
- Use parentheses to override precedence and ensure correct calculations
- Operations follow this order:
    1. Prefix increment/decrement (++x, --x)
    2. Positive/negative (+x, -x)
    3. Multiplication, division, modulus (*, /, %)
    4. Addition and subtraction (+, -). Use parentheses to override.
    5. Assignment (=, +=, etc.)



```
decimal price = 100m;
decimal discountPercent = .2m;
price = price * (1 - discountPercent);  // $80
```

Type Casting and Conversion

- .Net provides two type of casting. Implicit and explicit casting.
- Implicit casting: automatically converts smaller to larger types (e.g., int to double).
  - Can be used to convert data with a less precise type to more precise type. Also called a widening conversion.
- Explicit casting: requires a cast operator (e.g., double x = (double)myInt;).
  - Can be used to convert data with a more precise type to a less precise type. Also called narrowing conversion.
- Convert class methods: Convert.ToInt32(), Convert.ToDouble().
- Parsing methods: int.Parse(), double.Parse() convert string inputs to numbers.
- TryParse() safely handles invalid input without exceptions.



How implicit casting works

Casting from less precise to more precise data types

byte→short→int→long→decimal

int→double

short→float→double

char→int

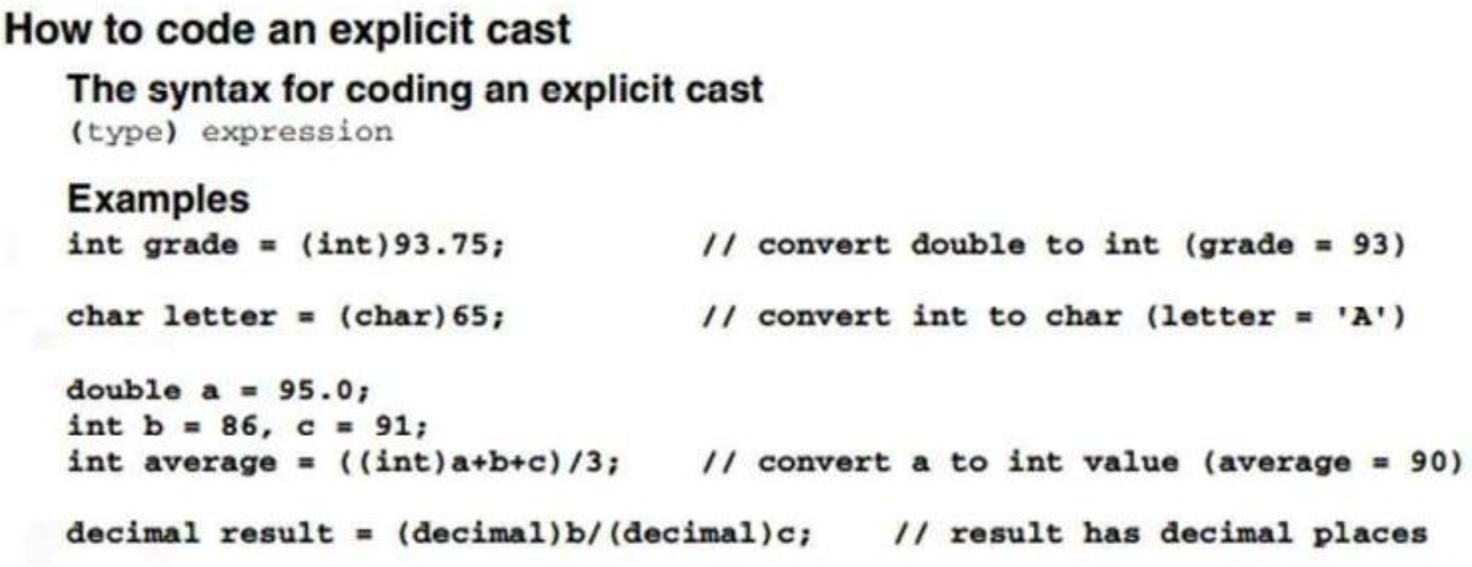
Examples

```
double grade = 93;                // convert int to double

int letter = 'A';                 // convert char to int

double a = 95.0;
int b = 86, c = 91;
double average = (a+b+c)/3;       // convert b and c to double values
                                  // (average = 90.666666...)
```

- Casting converts one data type to another explicitly.
- Use for precision in calculations, e.g., cast int to decimal for decimal division.
- Syntax: (type) expression



How to code an explicit cast

The syntax for coding an explicit cast

```
(type) expression
```

Examples

```
int grade = (int)93.75;           // convert double to int (grade = 93)

char letter = (char)65;           // convert int to char (letter = 'A')

double a = 95.0;
int b = 86, c = 91;
int average = ((int)a+b+c)/3;     // convert a to int value (average = 90)

decimal result = (decimal)b/(decimal)c;    // result has decimal places
```

Math calss:

- The Math class provides methods for common mathematical operations.
- Math is static class, no need to instantiate.

Method	Description
Abs	Absolute value
Ceiling	Rounds up to nearest integer
Floor	Rounds down to nearest integer
Max	Larger of two values
Min	Smaller of two values
Pow	Raises number to power
Round	Rounds to nearest integer or digits
Sqrt	Square root
Constants: **Math.PI, Math.E**	

Five static methods of the Math class

The syntax of the Round() method

```
Math.Round(decimalNumber[, precision[, mode]])
```

The syntax of the Pow() method

```
Math.Pow(number, power)
```

The syntax of the Sqrt() method

```
Math.Sqrt(number)
```

The syntax of the Min() and Max() methods

```
Math.{Min|Max}(number1, number2)
```

Statements that use static methods of the Math class

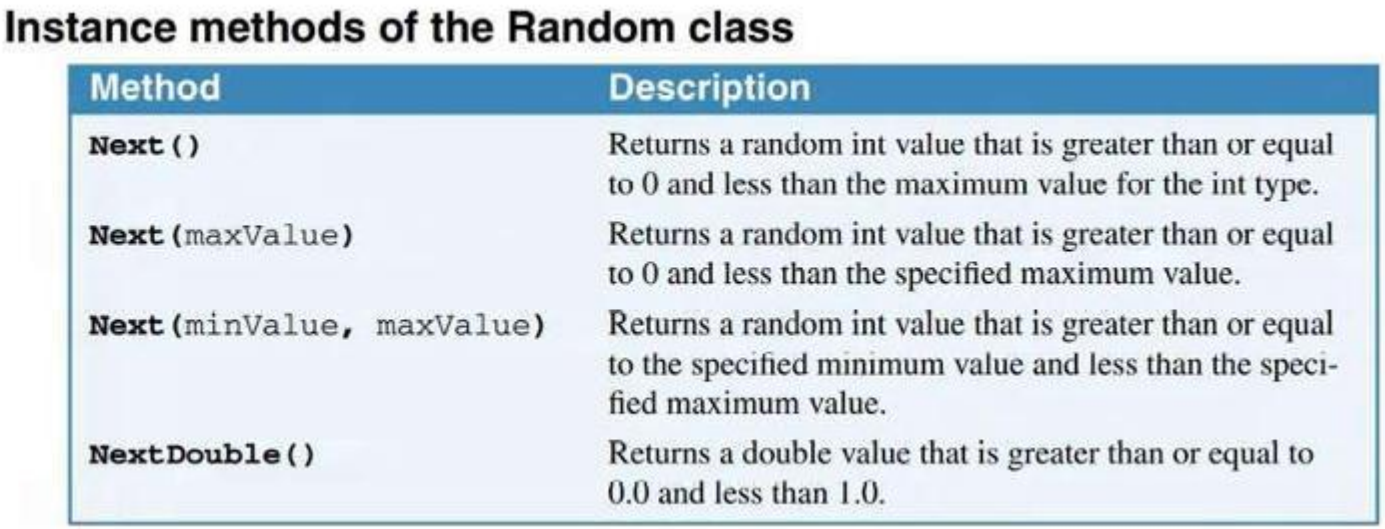
```
int shipWeight = Math.Round(shipWeightDouble);  // round to a whole number
double orderTotal = Math.Round(orderTotal, 2);  // round to 2 decimal places
double area = Math.Pow(radius, 2) * Math.PI;    // area of circle
double sqrtX = Math.Sqrt(x);
double maxSales = Math.Max(lastYearSales, thisYearSales);
int minQty = Math.Min(lastYearQty, thisYearQty);
```

Results from static methods of the Math class

Statement	Result	Statement	Result
`Math.Round(23.75, 1)`	23.8	`Math.Pow(5, 2)`	25
`Math.Round(23.85, 1)`	23.8	`Math.Sqrt(20.25)`	4.5
`Math.Round(23.744, 2)`	23.74	`Math.Max(23.75, 20.25)`	23.75
`Math.Round(23.745, 2)`	23.74	`Math.Min(23.75, 20.25)`	20.25
`Math.Round(23.745, 2, MidpointRounding.AwayFromZero)`			23.75

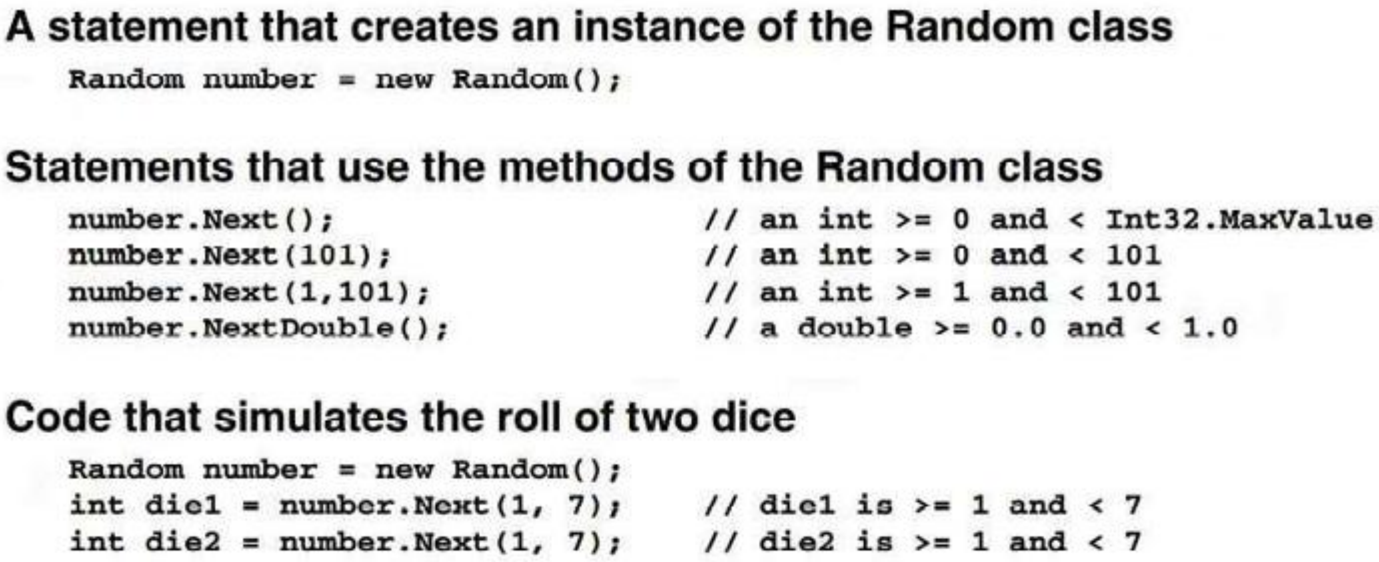
Random class

- Generate Random Numbers
- Use the Random class to generate pseudo-random numbers.
- Useful for games, simulations, etc.
- The methods of Random class are instance methods???.



Instance methods of the Random class

Method	Description
`Next()`	Returns a random int value that is greater than or equal to 0 and less than the maximum value for the int type.
`Next(maxValue)`	Returns a random int value that is greater than or equal to 0 and less than the specified maximum value.
`Next(minValue, maxValue)`	Returns a random int value that is greater than or equal to the specified minimum value and less than the specified maximum value.
`NextDouble()`	Returns a double value that is greater than or equal to 0.0 and less than 1.0.



A statement that creates an instance of the Random class

```
Random number = new Random();
```

Statements that use the methods of the Random class

```
number.Next();              // an int >= 0 and < Int32.MaxValue
number.Next(101);           // an int >= 0 and < 101
number.Next(1,101);         // an int >= 1 and < 101
number.NextDouble();        // a double >= 0.0 and < 1.0
```

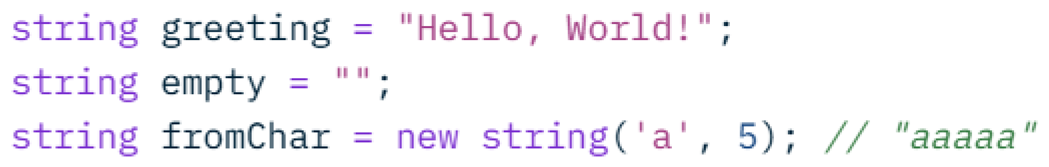
Code that simulates the roll of two dice

```
Random number = new Random();
int die1 = number.Next(1, 7);    // die1 is >= 1 and < 7
int die2 = number.Next(1, 7);    // die2 is >= 1 and < 7
```

- The **String class** - How to Work with Strings
    - Strings are immutable sequences of characters.
    - Declare with double quotes.

Declaring and Initializing Strings

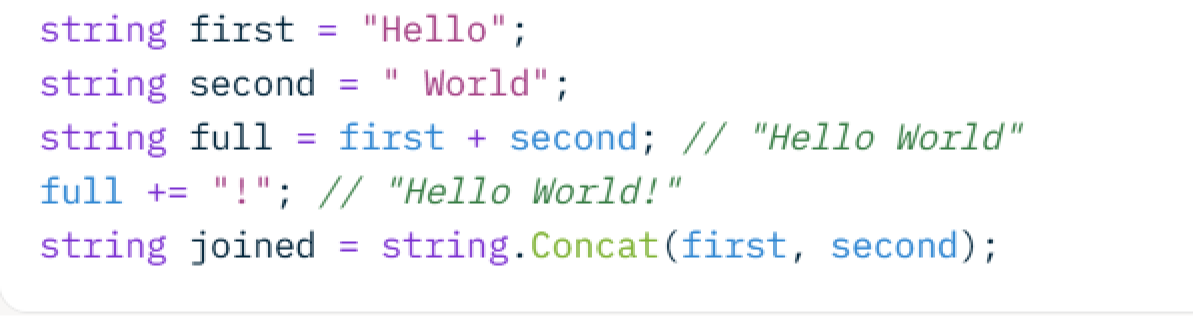
- Syntax: string name = "value";



```
string greeting = "Hello, World!";
string empty = "";
string fromChar = new string('a', 5); // "aaaaa"
```

Joining and Appending Strings

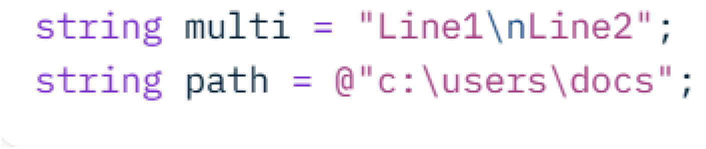
- Join: + operator or string.Concat
- Append: +=



```
string first = "Hello";
string second = " World";
string full = first + second; // "Hello World"
full += "!"; // "Hello World!"
string joined = string.Concat(first, second);
```

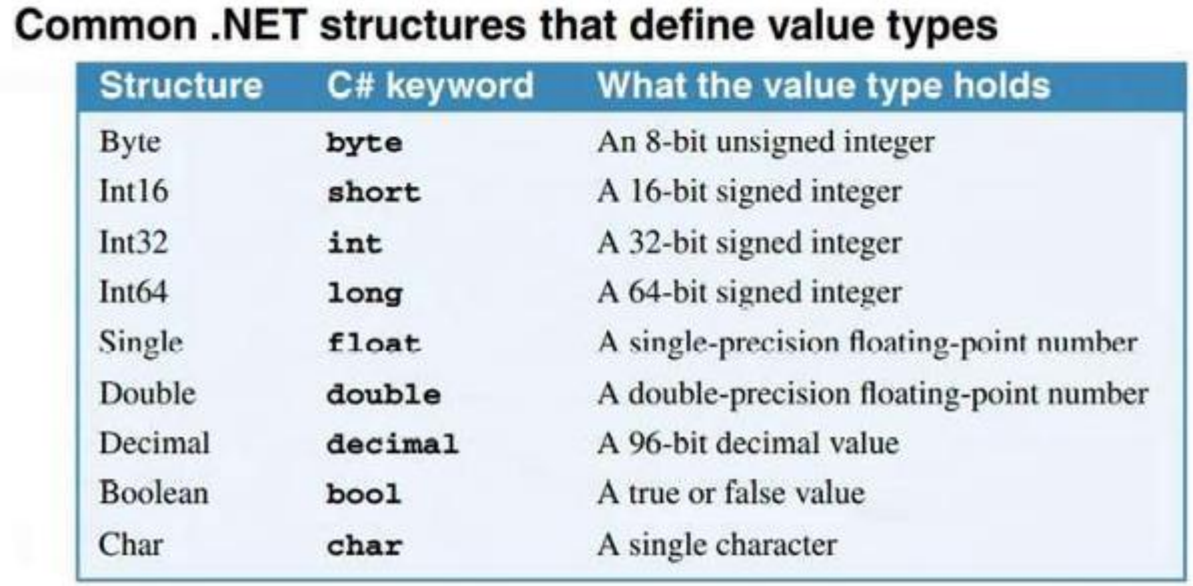
Including Special Characters in Strings

- Use escape sequences: \n (newline), \t (tab), " (quote), \ (backslash)
- Verbatim strings: @"c:\path" (ignores escapes)



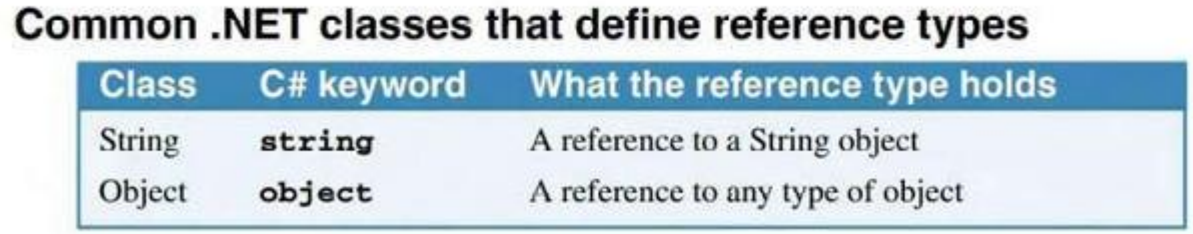
```
string multi = "Line1\nLine2";
string path = @"c:\users\docs";
```

- **How to Convert Data Types**
  - Use methods to convert between types, handle potential errors.
  - **.NET Structures for Data Types**
    - Each value type has a structure: Int32 for int, Double for double, etc.
    - Methods like Parse, TryParse.



Common .NET structures that define value types

Structure	C# keyword	What the value type holds
Byte	`byte`	An 8-bit unsigned integer
Int16	`short`	A 16-bit signed integer
Int32	`int`	A 32-bit signed integer
Int64	`long`	A 64-bit signed integer
Single	`float`	A single-precision floating-point number
Double	`double`	A double-precision floating-point number
Decimal	`decimal`	A 96-bit decimal value
Boolean	`bool`	A true or false value
Char	`char`	A single character



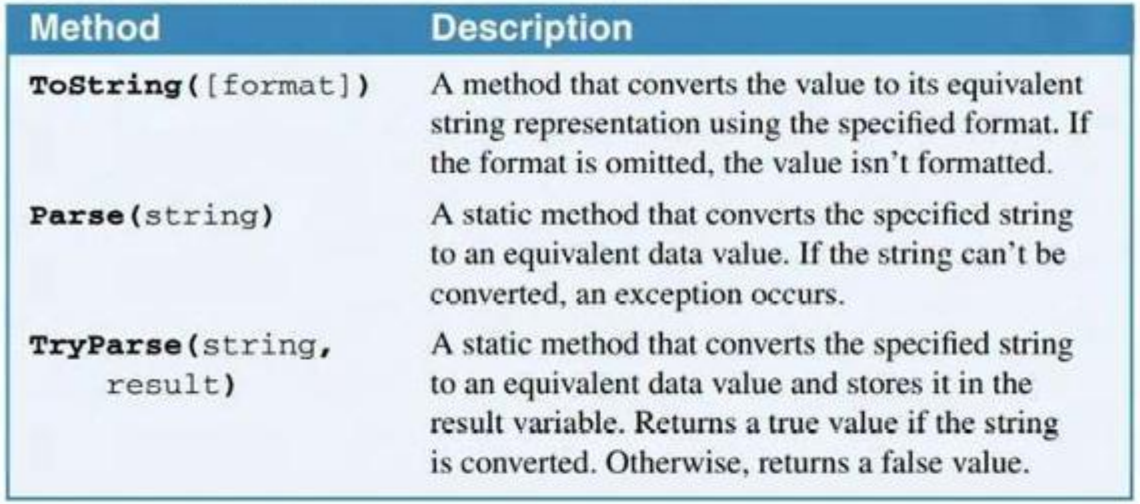
Common .NET classes that define reference types

Class	C# keyword	What the reference type holds
String	`string`	A reference to a String object
Object	`object`	A reference to any type of object

Methods to Convert Data Types

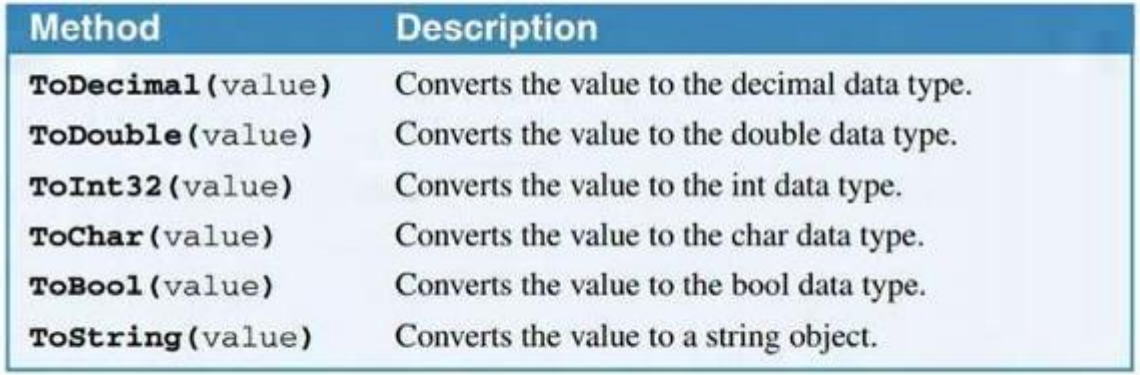
- Convert.ToType(value)
- type.Parse(string)
- type.TryParse(string, out var)

Common methods for data conversion

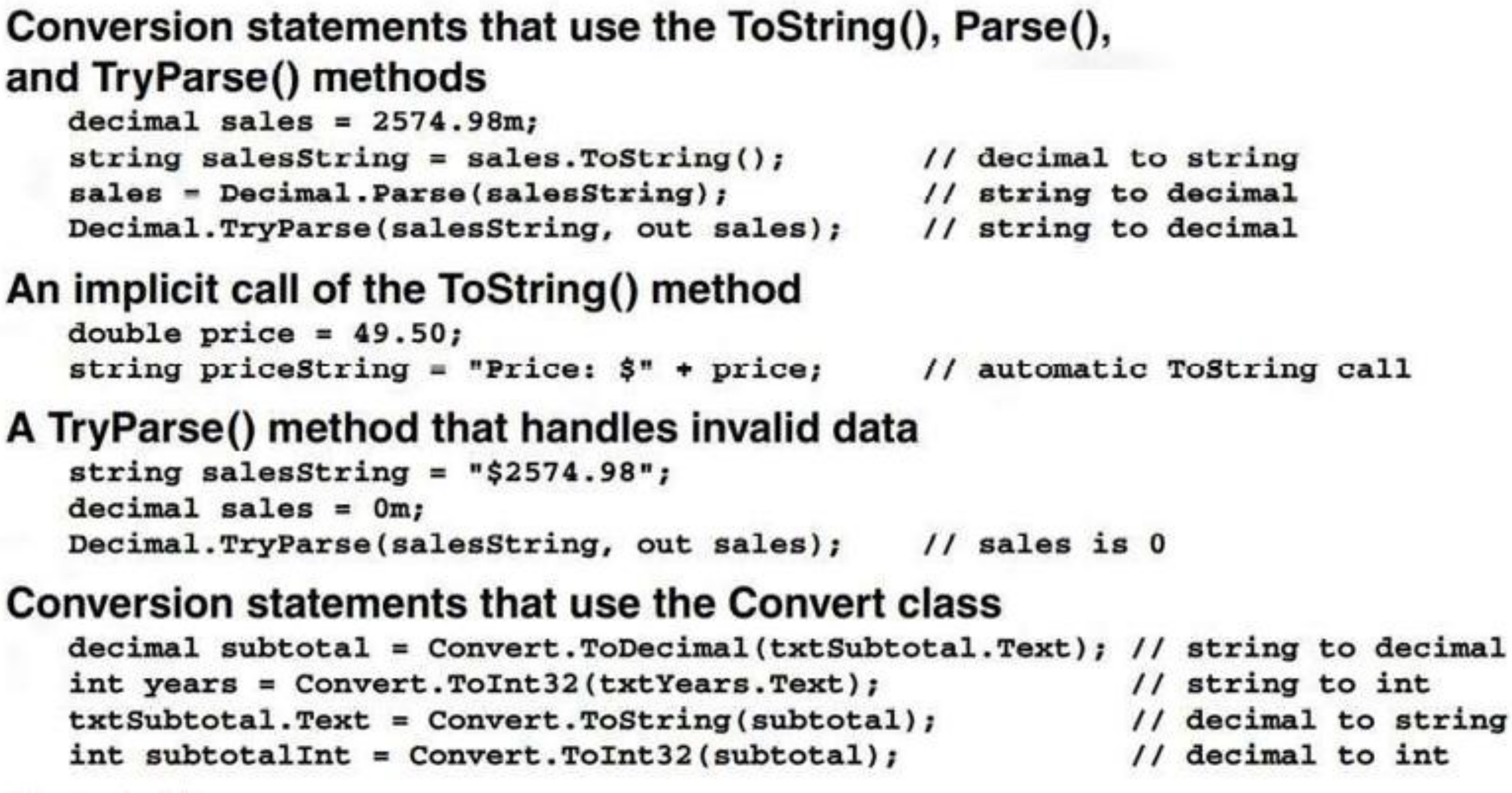


Method	Description
`ToString([format])`	A method that converts the value to its equivalent string representation using the specified format. If the format is omitted, the value isn't formatted.
`Parse(string)`	A static method that converts the specified string to an equivalent data value. If the string can't be converted, an exception occurs.
`TryParse(string, result)`	A static method that converts the specified string to an equivalent data value and stores it in the result variable. Returns a true value if the string is converted. Otherwise, returns a false value.

Some of the static methods of the Convert class



Method	Description
`ToDecimal(value)`	Converts the value to the decimal data type.
`ToDouble(value)`	Converts the value to the double data type.
`ToInt32(value)`	Converts the value to the int data type.
`ToChar(value)`	Converts the value to the char data type.
`ToBool(value)`	Converts the value to the bool data type.
`ToString(value)`	Converts the value to a string object.



Conversion statements that use the ToString(), Parse(), and TryParse() methods

```
decimal sales = 2574.98m;
string salesString = sales.ToString();       // decimal to string
sales = Decimal.Parse(salesString);          // string to decimal
Decimal.TryParse(salesString, out sales);    // string to decimal
```

An implicit call of the ToString() method

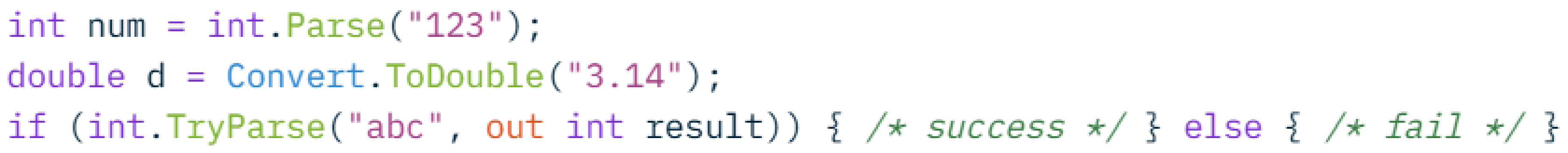
```
double price = 49.50;
string priceString = "Price: $" + price;     // automatic ToString call
```

A TryParse() method that handles invalid data

```
string salesString = "$2574.98";
decimal sales = 0m;
Decimal.TryParse(salesString, out sales);    // sales is 0
```

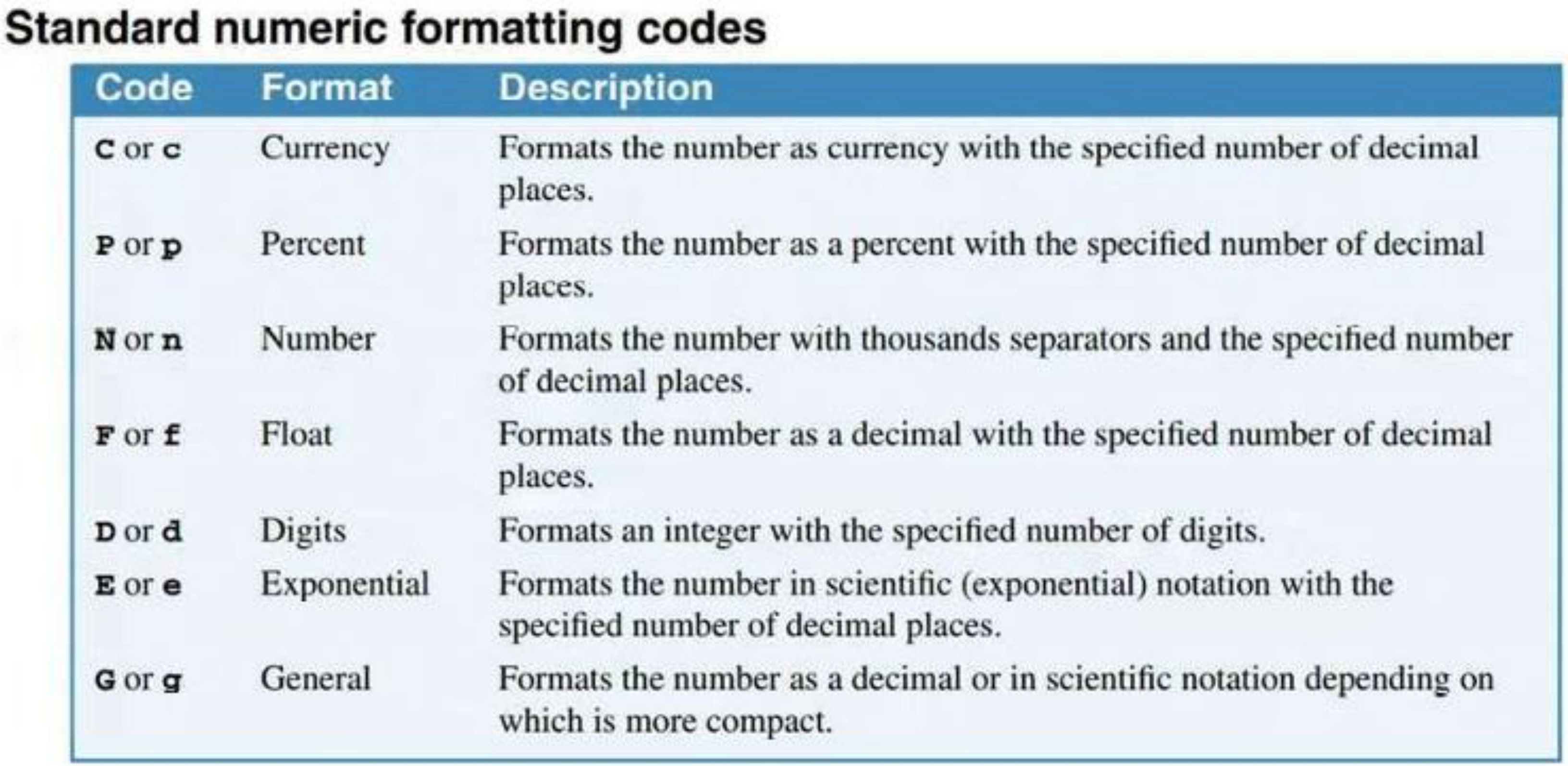
Conversion statements that use the Convert class

```
decimal subtotal = Convert.ToDecimal(txtSubtotal.Text); // string to decimal
int years = Convert.ToInt32(txtYears.Text);             // string to int
txtSubtotal.Text = Convert.ToString(subtotal);          // decimal to string
int subtotalInt = Convert.ToInt32(subtotal);            // decimal to int
```



```
int num = int.Parse("123");
double d = Convert.ToDouble("3.14");
if (int.TryParse("abc", out int result)) { /* success */ } else { /* fail */ }
```

- Converting Numbers to Formatted Strings



Standard numeric formatting codes

Code	Format	Description
`C` or `c`	Currency	Formats the number as currency with the specified number of decimal places.
`P` or `p`	Percent	Formats the number as a percent with the specified number of decimal places.
`N` or `n`	Number	Formats the number with thousands separators and the specified number of decimal places.
`F` or `f`	Float	Formats the number as a decimal with the specified number of decimal places.
`D` or `d`	Digits	Formats an integer with the specified number of digits.
`E` or `e`	Exponential	Formats the number in scientific (exponential) notation with the specified number of decimal places.
`G` or `g`	General	Formats the number as a decimal or in scientific notation depending on which is more compact.

How to use the ToString() method to format a number

Statement	Example
`string monthlyAmount = amount.ToString("c");`	$1,547.20
`string interestRate = interest.ToString("p1");`	2.3%
`string quantityString = quantity.ToString("n0");`	15,000
`string paymentString = payment.ToString("f3");`	432.818

How to use the Format() method of the String class to format a number

Statement	Result
`string monthlyAmount = String.Format("{0:c}", 1547.2m);`	$1,547.20
`string interestRate = String.Format("{0:p1}", .023m);`	2.3%
`string quantityString = String.Format("{0:n0}", 15000);`	15,000
`string paymentString = String.Format("{0:f3}", 432.8175);`	432.818

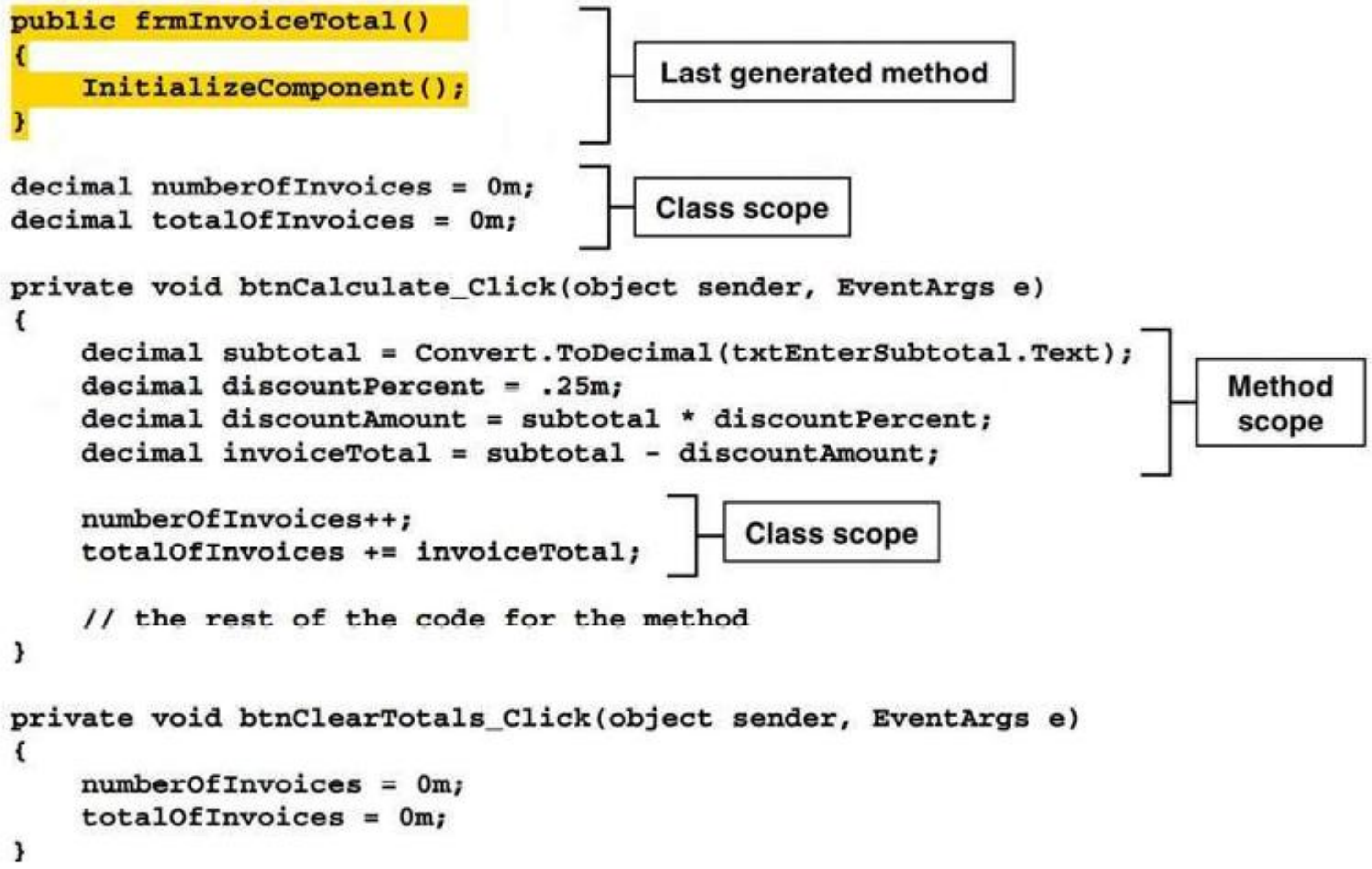
The syntax of the format specification used by the Format() method

```
{index:formatCode}
```

Additional points for Working with Data

- Scope: Visibility of variables.
- Enumerations: Named constants.
- Nullable types: Value types that can be null.

Code that declares and uses variables with class scope



```
public frmInvoiceTotal()
{
    InitializeComponent();
}

decimal numberOfInvoices = 0m;
decimal totalOfInvoices = 0m;

private void btnCalculate_Click(object sender, EventArgs e)
{
    decimal subtotal = Convert.ToDecimal(txtEnterSubtotal.Text);
    decimal discountPercent = .25m;
    decimal discountAmount = subtotal * discountPercent;
    decimal invoiceTotal = subtotal - discountAmount;

    numberOfInvoices++;
    totalOfInvoices += invoiceTotal;

    // the rest of the code for the method
}

private void btnClearTotals_Click(object sender, EventArgs e)
{
    numberOfInvoices = 0m;
    totalOfInvoices = 0m;
}
```

How to Declare and Use Enumerations

- An enumeration is a set of related constants that define a value type where each constant is known as a member of the enumeration. The enumerations provided by the .NET Framework are generally used to set object properties and to specify the values that are passed to methods.
- For example, the FormBorderStyle enumeration includes a group of constants that you can use to specify the settings for the FormBorderStyle property of a form.
- E.g
  - enum Type { Value1, Value2 }
  - Underlying int by default.

Some of the constants in the FormBorderStyle enumeration

Constant	Description
`FormBorderStyle.FixedDialog`	A fixed, thick border typically used for dialog boxes.
`FormBorderStyle.FixedSingle`	A single-line border that isn't resizable.
`FormBorderStyle.Sizable`	A resizable border

A statement that uses the FormBorderStyle enumeration

```
this.FormBorderStyle = FormBorderStyle.FixedSingle;
```

The syntax for declaring an enumeration

```
enum EnumerationName [: type]
{
    ConstantName1 [= value][,
    ConstantName2 [= value]]...
}
```

An enumeration that sets the constant values to 0, 1, and 2

```
enum Terms
{
    Net30Days,
    Net60Days,
    Net90Days
}
```

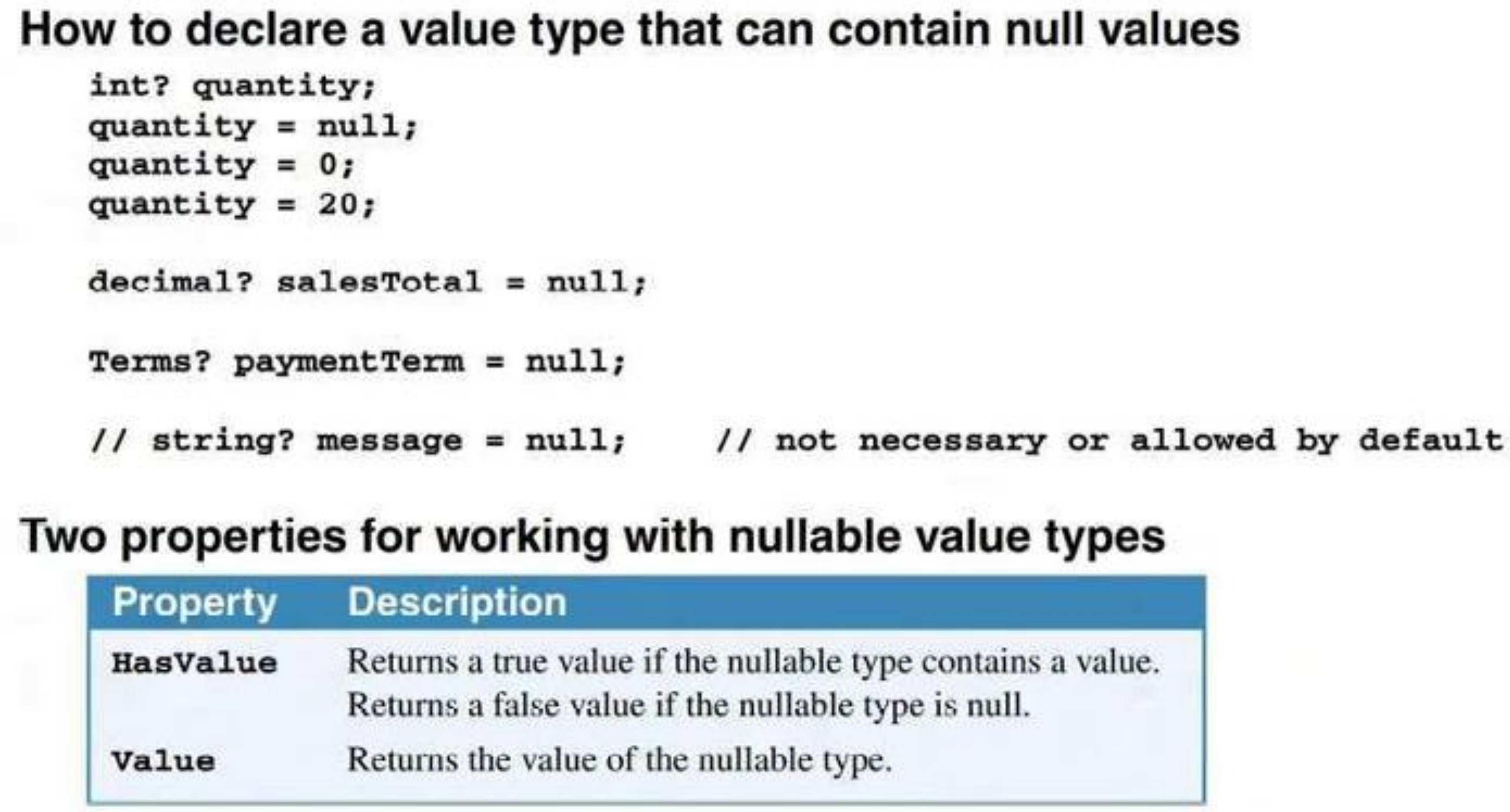
An enumeration that sets the constant values to 30, 60, and 90

```
enum TermValues : short
{
    Net30Days = 30,
    Net60Days = 60,
    Net90Days = 90
}
```

Statements that use the constants in these enumerations

```
Terms t = Terms.Net30Days;
int i = (int) Terms.Net30Days;             // i is 0
int i = (int) TermValues.Net60Days;        // i is 60
string s = Terms.Net30Days.ToString();     // s is "Net30Days"
```

Nullable types: Value types that can be null.



How to declare a value type that can contain null values

```
int? quantity;
quantity = null;
quantity = 0;
quantity = 20;

decimal? salesTotal = null;

Terms? paymentTerm = null;

// string? message = null;    // not necessary or allowed by default
```

Two properties for working with nullable value types

Property	Description
`HasValue`	Returns a true value if the nullable type contains a value. Returns a false value if the nullable type is null.
`Value`	Returns the value of the nullable type.

How to use the properties of a nullable value type

```
if (quantity.HasValue) {
    int qty = quantity.Value;
}
```

How to use the null-coalescing operator to assign a default value

```
int qty = quantity ?? -1;
```

How to use the null-coalescing assignment operator

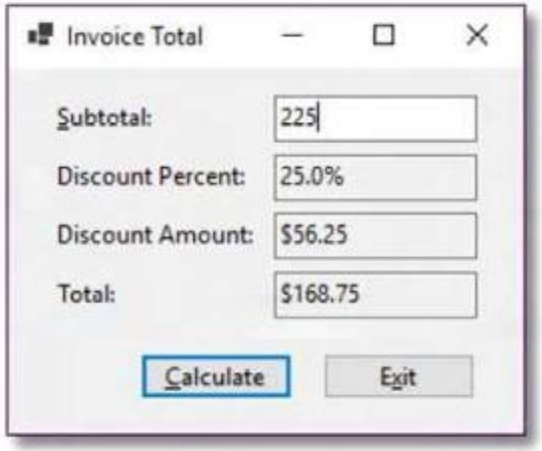
```
salesTotal ??= 0.0m;
```

How to use nullable value types in arithmetic expressions

```
decimal? sales1 = 3267.58m;
decimal? sales2 = null;
decimal? salesTotal = sales1 + sales2;    // result = null
```

Exercise: The Invoice Total form

The Invoice Total form



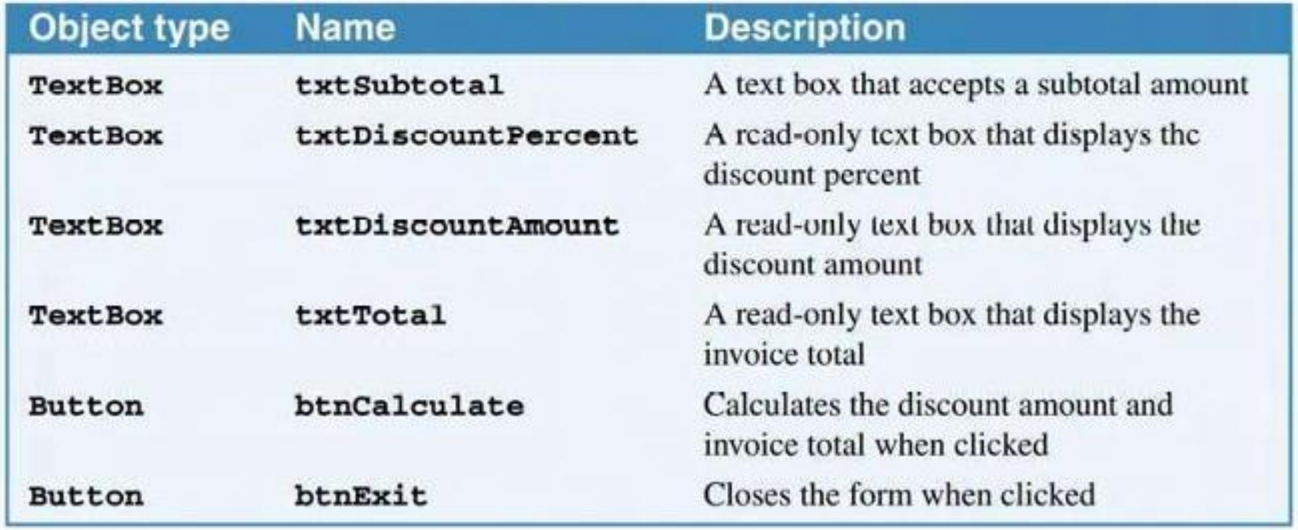
Subtotal: 225

Discount Percent: 25.0%

Discount Amount: $56.25

Total: $168.75

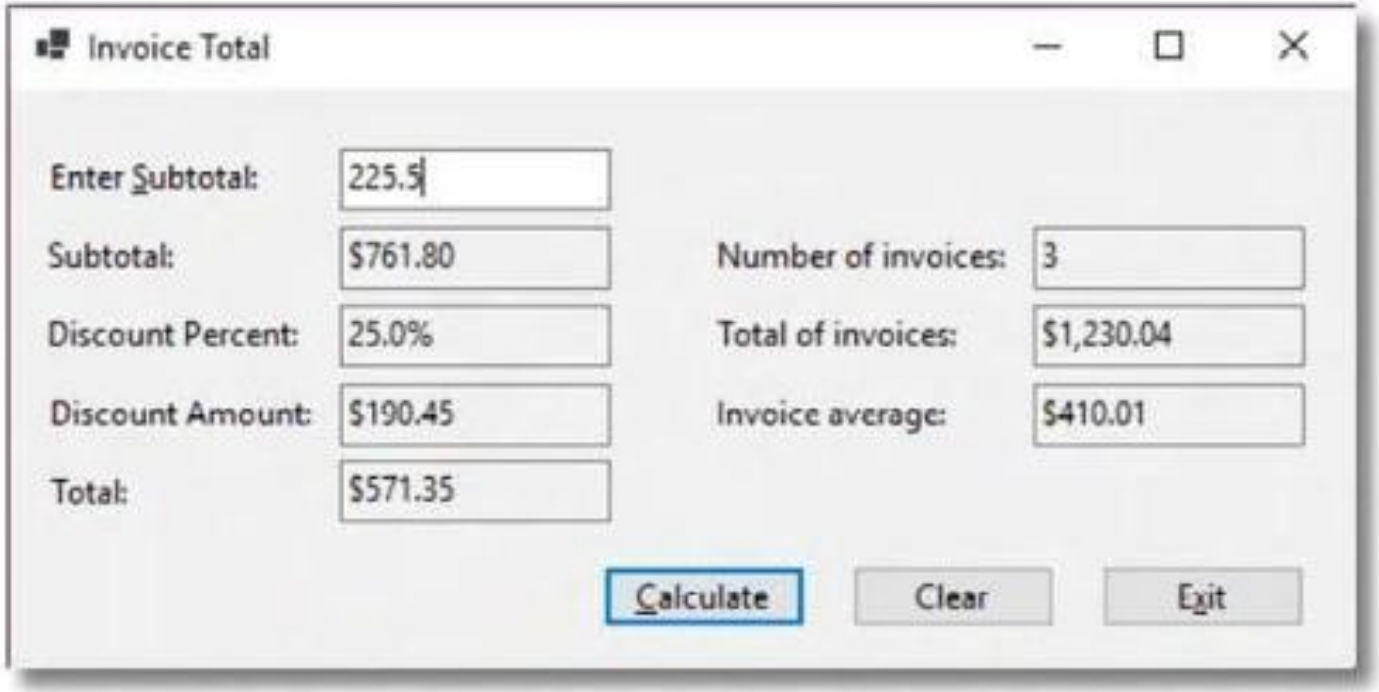
The controls that are referred to in the code



Object type	Name	Description
TextBox	txtSubtotal	A text box that accepts a subtotal amount
TextBox	txtDiscountPercent	A read-only text box that displays the discount percent
TextBox	txtDiscountAmount	A read-only text box that displays the discount amount
TextBox	txtTotal	A read-only text box that displays the invoice total
Button	btnCalculate	Calculates the discount amount and invoice total when clicked
Button	btnExit	Closes the form when clicked

The Invoice Total form

The enhanced Invoice Total form



Invoice Total

Enter Subtotal:	225.5		
Subtotal:	$761.80	Number of invoices:	3
Discount Percent:	25.0%	Total of invoices:	$1,230.04
Discount Amount:	$190.45	Invoice average:	$410.01
Total:	$571.35		

End of Review of C# basics (variables, data types)

Next - control structures; Delegates and event handlers; Event-driven design patterns; Working with arrays, lists, and strings; Applying OOP principles (classes, inheritance) in GUI apps