Advanced Programming

Lecture 02: Introduction to C# Apps

Some slides are borrowed from Dr. Shahriar Bijani's slides

A Simple C# Program

```
1     // Fig. 3.1: Welcome1.cs
2     // A first program in C#.
3
4     using System;
5
6     class Welcome1
7     {
8        static void Main( string[] args )
9        {
10          Console.WriteLine( "Welcome to C# Programming!" );
11       } // end of Main
12 } // end of class Welcome1
```

- Program output:

```
Welcome to C# Programming!
```

A Simple C# Program

- **Comments**
  - **Single-line comment:** `//`
  - **Multi-line comments:** `/*  */`

```
/*  a multiline
      comment example */
```

  - Comments are ignored by the compiler
  - Used only for human readers

A Simple C# Program

- **Namespaces**
  - named collections of related classes
  - Allows the easy reuse of code
  - Many namespaces are found in the .NET framework library
  - **using directive** tells the compiler where to look for a used class in this app

Error-Prevention Tip 3.1

- Forgetting to include a ***using*** directive for a namespace of a used class typically results in a compilation error: "The name 'Console' does not exist in the current context."

Simple Program: Printing a line of text

- Keywords
  - Words that cannot be used as **identifier** (variable/class/… names)
  - All keywords are lowercase

- Classes
  - Class names can only be one word long (i.e. no white space)
  - Each class name is an **identifier:**
    - Can contain letters, digits, and underscores (_)
    - Cannot start with digits
    - Can start with the at symbol (@)

Keywords and contextual keywords

abstract	as	base	bool	break	byte
case	catch	char	checked	class	const
continue	decimal	default	delegate	do	double
else	enum	event	explicit	extern	false
finally	fixed	float	for	foreach	goto
if	implicit	in	int	interface	internal
is	lock	long	namespace	new	null
object	operator	out	override	params	private
protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string
struct	switch	this	throw	true	try
typeof	uint	ulong	unchecked	unsafe	ushort
using	virtual	void	volatile	while	

Contextual Keywords

add	alias	ascending	async	await	by
descending	dynamic	equals	from	get	global
group	into	join	let	nameof	on
orderby	partial	remove	select	set	value
var	where	yield			

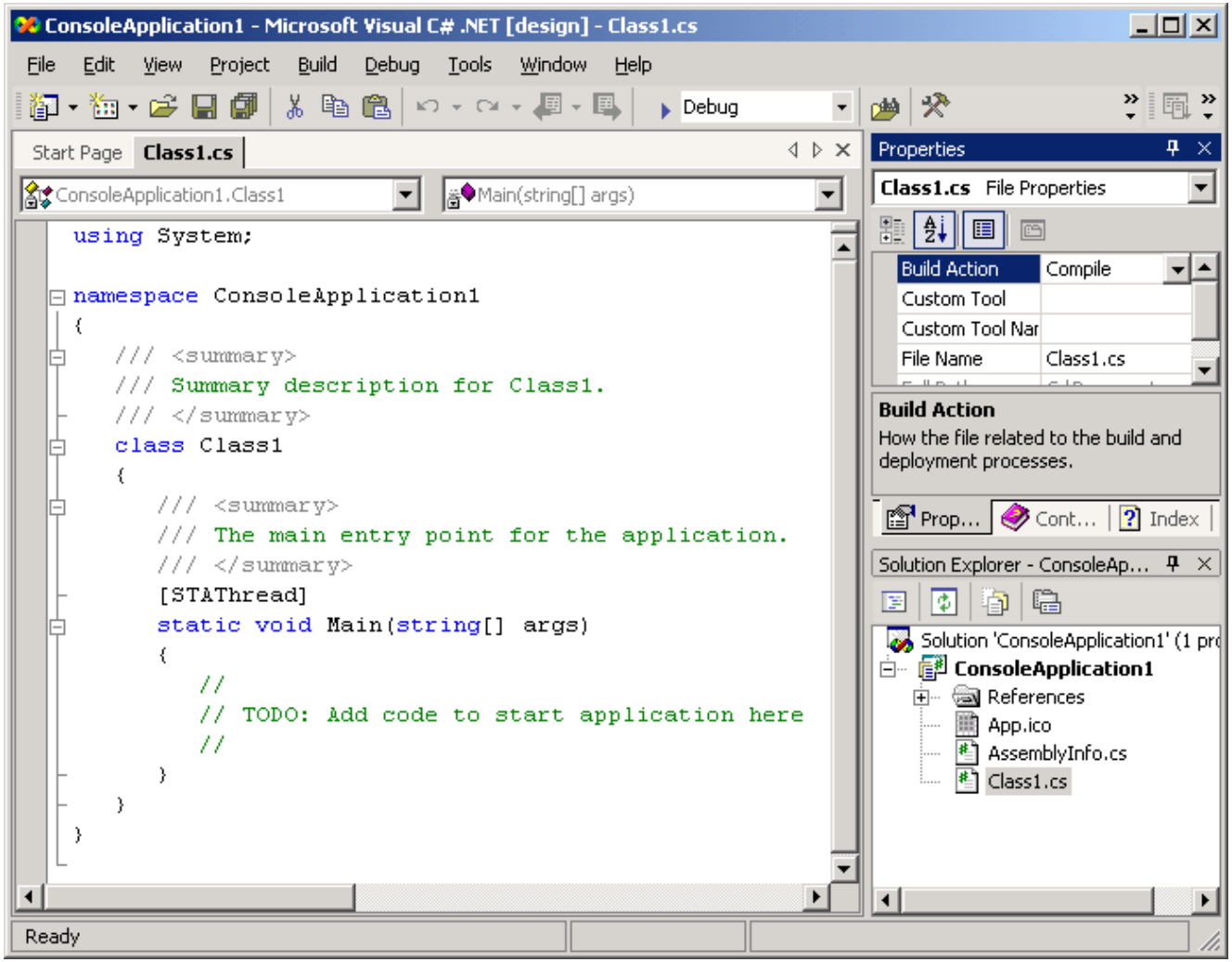
Simple Program: Printing a line of text

- **Methods**
  - Building blocks of programs
  - The `Main` method
    - Each console or windows application must have exactly one
    - All programs start by executing the `Main` method
  - Braces are used to start ({) and end (}) a method

Simple Program: Printing a line of text

- **Graphical User Interface (GUI)**
  - GUIs: to easily **get data** from the user/ **display data** to the user
  - **Message boxes**
    - Within the **`System.Windows.Forms`** namespace
    - Used to prompt or display information to the user

3.2 Simple Program: Printing a Line of Text



Visual Studio .NET-generated console application.

Program Output

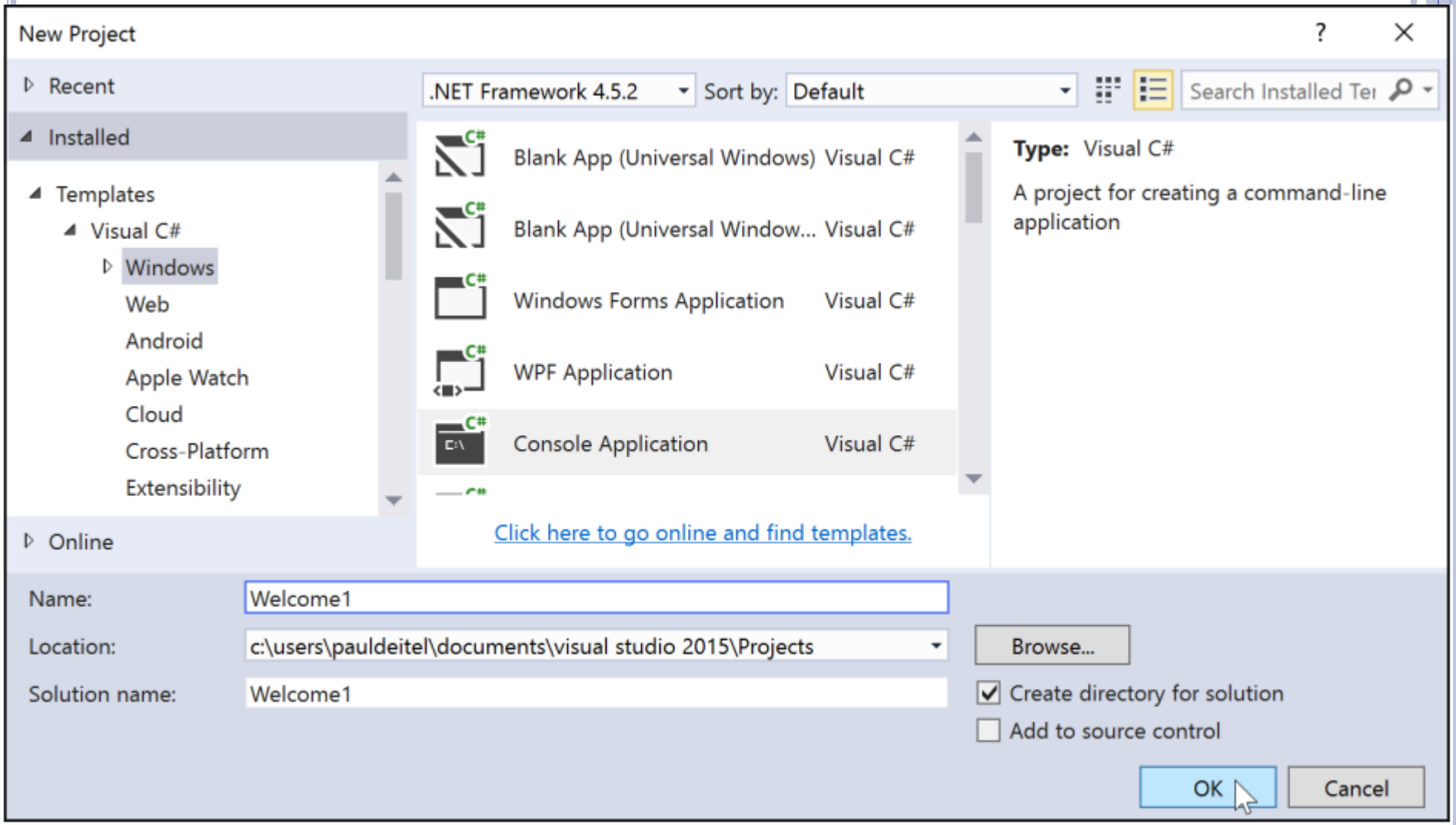
```
1    // Fig. 3.4: Welcome2.cs
2    // Printing a line with multiple statements.
3
4    using System;
5
6    class Welcome2
7    {
8       static void Main( string[] args )
9       {
10         Console.Write( "Welcome to " );
11         Console.WriteLine( "C# Programming!" );
12      }
13   }
```

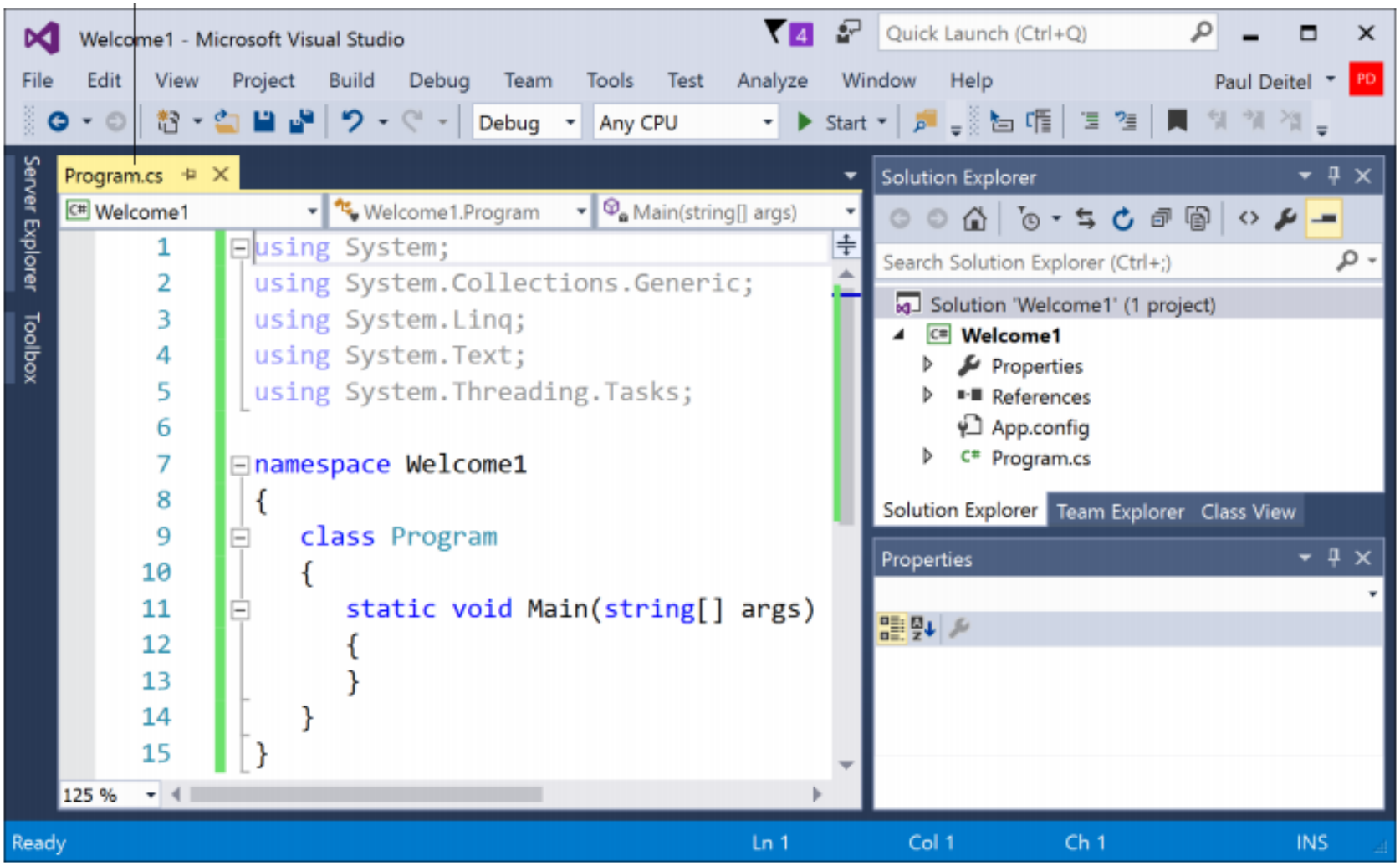
Simple Program: Printing a Line of Text

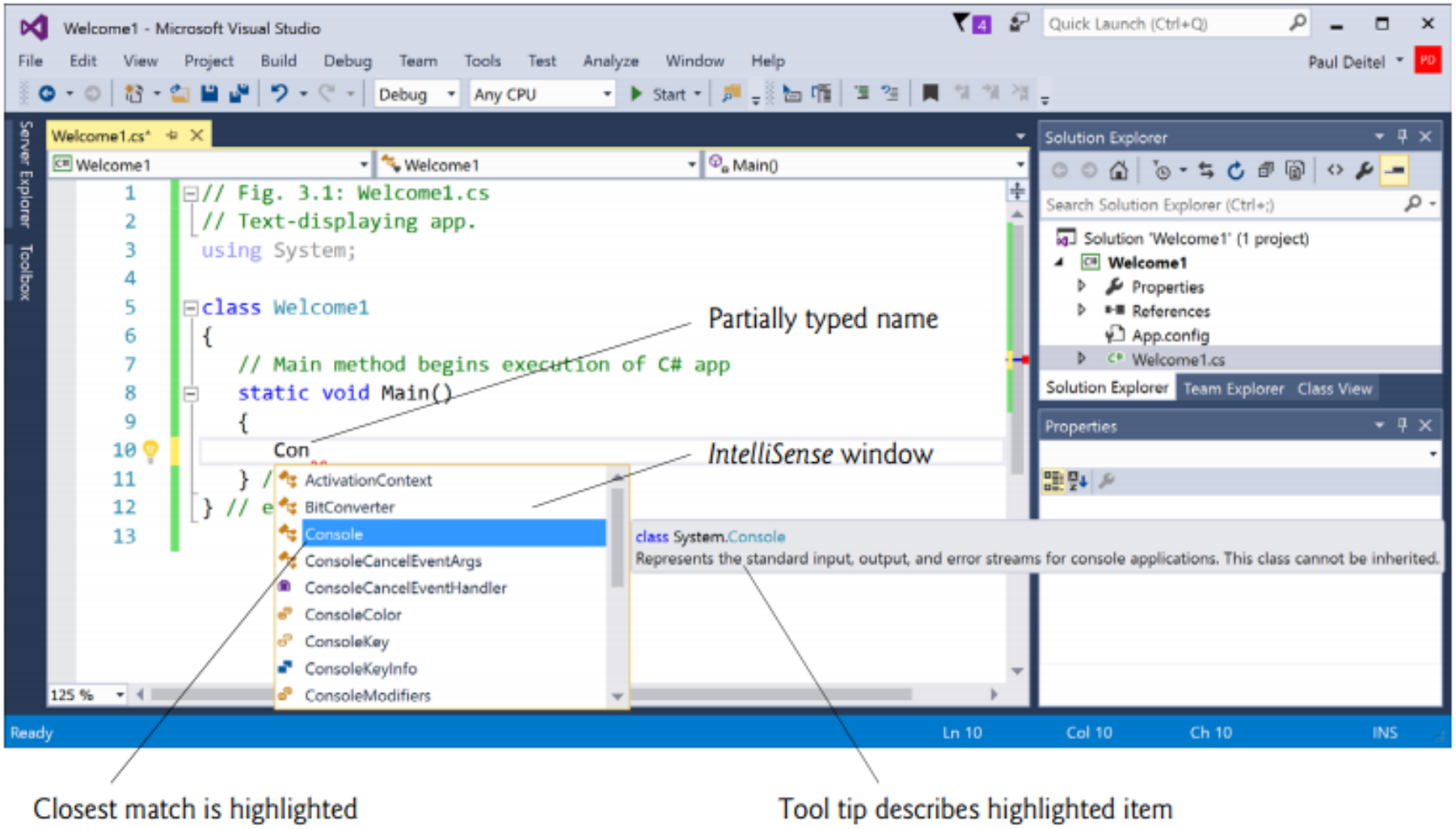
Escape sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the previous characters output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote (") character.

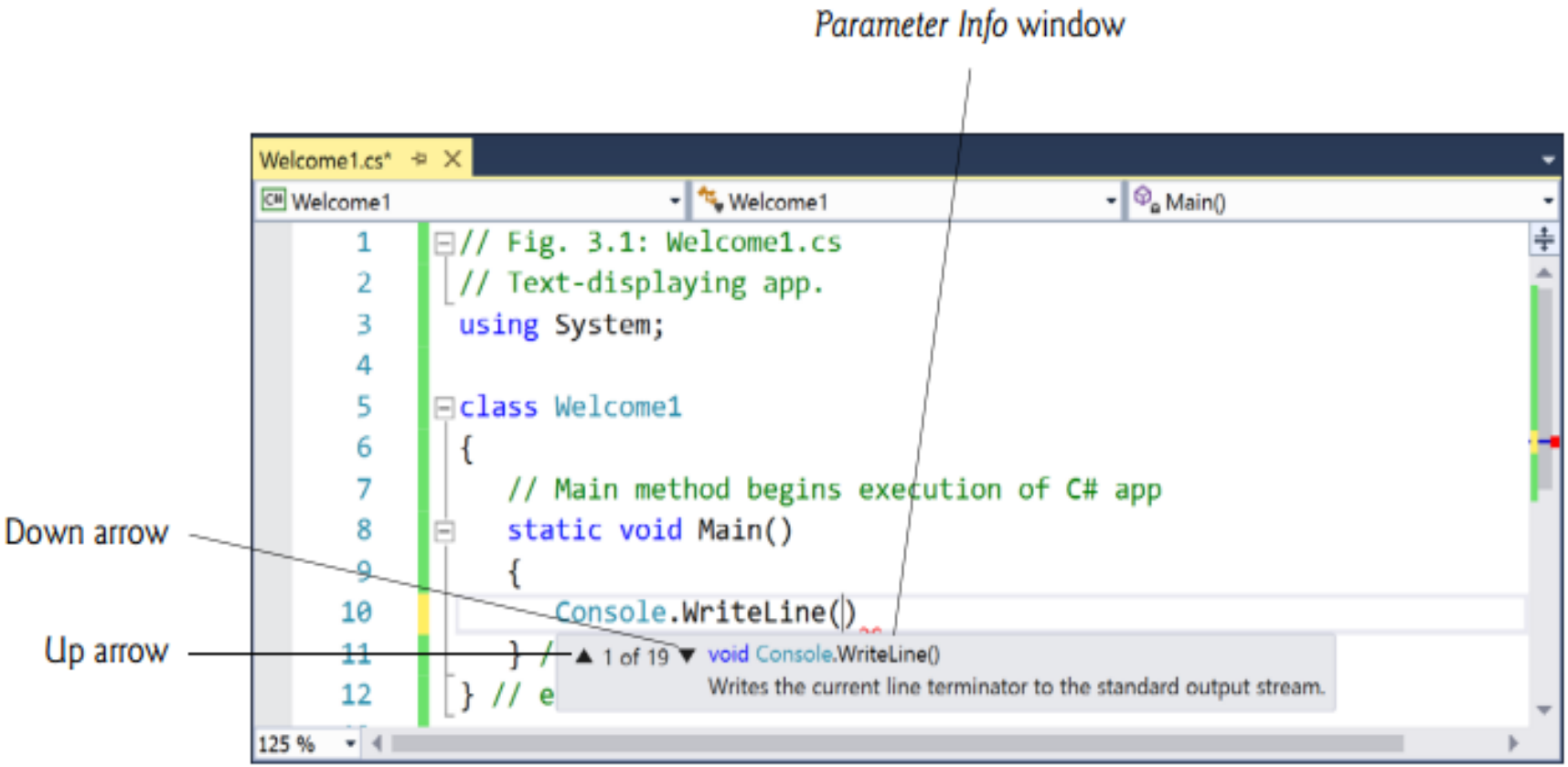
Fig. 3.6 Some common escape sequences.

Creating a console app




Welcome1 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Paul Deitel
Debug
Any CPU
Start
Server Explorer
Toolbox
Program.cs
Welcome1
Welcome1.Program
Main(string[] args)
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace Welcome1
8 {
9 class Program
10 {
11 static void Main(string[] args)
12 {
13 }
14 }
15 }
125 %
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Welcome1' (1 project)
Welcome1
Properties
References
App.config
Program.cs
Solution Explorer
Team Explorer
Class View
Properties
Ready
Ln 1
Col 1
Ch 1
INS


Welcome1 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Paul Deitel
Debug
Any CPU
Start
Welcome1.cs*
Welcome1
Welcome1
Main()
1 // Fig. 3.1: Welcome1.cs
2 // Text-displaying app.
3 using System;
4
5 class Welcome1
6 {
7 // Main method begins execution of C# app
8 static void Main()
9 {
10 Con
11 } /
12 } // e
13
Partially typed name
IntelliSense window
ActivationContext
BitConverter
Console
ConsoleCancelEventArgs
ConsoleCancelEventHandler
ConsoleColor
ConsoleKey
ConsoleKeyInfo
ConsoleModifiers
class System.Console
Represents the standard input, output, and error streams for console applications. This class cannot be inherited.
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Welcome1' (1 project)
Welcome1
Properties
References
App.config
Welcome1.cs
Solution Explorer Team Explorer Class View
Properties
Server Explorer
Toolbox
125 %
Ready
Ln 10
Col 10
Ch 10
INS
Closest match is highlighted
Tool tip describes highlighted item


Parameter Info window
Down arrow
Up arrow
Welcome1.cs*
Welcome1
Welcome1
Main()
1 // Fig. 3.1: Welcome1.cs
2 // Text-displaying app.
3 using System;
4
5 class Welcome1
6 {
7 // Main method begins execution of C# app
8 static void Main()
9 {
10 Console.WriteLine()
11 } /
12 } // e
1 of 19
void Console.WriteLine()
Writes the current line terminator to the standard output stream.
125 %

Compiling and running

```
1    // Fig. 3.5: Welcome3.cs
2    // Printing multiple lines with a single statement.
3
4    using System;
5
6    class Welcome3
7    {
8       static void Main( string[] args )
9       {
10        Console.WriteLine("Welcome\nto\nC#\nProgramming!");
11      }
12   }
```

```
Welcome
to
C#
Programming!
```

COMPILING AND RUNNING

```
1    // Fig. 3.7: Welcome4.cs
2    // Printing multiple lines in a dialog Box.
3
4    using System;
5    using System.Windows.Forms;
6
7    class Welcome4
8    {
9       static void Main( string[] args )
10      {
11        MessageBox.Show("Welcome\nto\nC#\nprogramming!");
12      }
13   }
```

Simple Program2: Adding Integers

- **Primitive data types**
  - Data types that are built into C#
    - String, Int, Double, Char, Long
    - 15 primitive data types (chapter 4)
  - Each data type name is a C# keyword
  - Same type variables can be declared on separate lines or on one line
- **Console.ReadLine()**
  - to get a value from the user input
- **Int32.Parse()**
  - to convert a string argument to an integer
  - to do math

Addition.cs

```
1   // Fig. 3.11: Addition.cs  - An addition program.
2   using System;
3   class Addition
4   {
5      static void Main( string[] args )
6      {
7        string firstNumber,   // first string entered by user
8               secondNumber;  // second string entered by user
9        int number1,          // first number to add
10            number2,          // second number to add
11            sum;              // sum of number1 and number2
12        // prompt for and read first number from user as string
13        Console.Write( "Please enter the first integer: " );
14        firstNumber = Console.ReadLine();
15        // read second number from user as string
16        Console.Write( "\nPlease enter the second integer: " );
17        secondNumber = Console.ReadLine();
18        // convert numbers from type string to type int
19        number1 = Int32.Parse( firstNumber );
20        number2 = Int32.Parse( secondNumber );
21        // add numbers
22        sum = number1 + number2;
23        Console.WriteLine( "\nThe sum is {0}.", sum );
24     }
25  }
```

Arithmetic

- Arithmetic operations
  - Not all operations use the same symbol
    - Asterisk (*) is multiplication
    - Slash (/) is division
    - Percent sign (%) is the modulus operator
    - Plus (+) and minus (-) are the same
  - There are no exponents
- Division
  - Division can vary depending on the variables used
    - When dividing two integers the result is always rounded down to an integer

Arithmetic

- Order
  - Parenthesis are done first
  - Division, multiplication and modulus are done second
    - Left to right
  - Addition and subtraction are done last
    - Left to right

ARITHMETIC

C# operation	Arithmetic operator	Algebraic expression	C# expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	$b*m$	`b * m`
Division	/	x / y	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Fig. 3.15 Arithmetic operators.

ARITHMETIC

Operator(s)	Operation	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/` or `%`	Multiplication Division Modulus	Evaluated second. If there are several such operators, they are evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several such operators, they are evaluated left to right.

Fig. 3.16 Precedence of arithmetic operators.

3Decision Making: Equality and Relational Operators

- The `if` structure
  - Used to make a decision based on the truth of the condition
    - True: a statement is performed
    - False: the statement is skipped over
  - Fig. 3.18 lists the equality and rational operators
    - There should be no spaces separating the operators

3.6 Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C# equality or relational operator	Example of C# condition	Meaning of C# condition
Equality operators			
=	==	`x == y`	`x` is equal to `y`
≠	!=	`x != y`	`x` is not equal to `y`
Relational operators			
>	>	`x > y`	`x` is greater than `y`
<	<	`x < y`	`x` is less than `y`
≥	>=	`x >= y`	`x` is greater than or equal to `y`
≤	<=	`x <= y`	`x` is less than or equal to `y`

Fig. 3.18 Equality and relational operators.

Comparison.cs

```
1  // Fig. 3.19: Comparison.cs
2  // Using if statements, relational operators and equality
3  // operators.
4
5  using System;
6
7  class Comparison
8  {
9     static void Main( string[] args )
10    {
11       int number1,          // first number to compare
12           number2;          // second number to compare
13
14       // read in first number from user
15       Console.Write( "Please enter first integer: " );
16       number1 = Int32.Parse( Console.ReadLine() );
17
18       // read in second number from user
19       Console.Write( "\nPlease enter second integer: " );
20       number2 = Int32.Parse( Console.ReadLine() );
21
22       if ( number1 == number2 )
23          Console.WriteLine( number1 + " == " + number2 );
24
25       if ( number1 != number2 )
26          Console.WriteLine( number1 + " != " + number2 );
27
28       if ( number1 < number2 )
29          Console.WriteLine( number1 + " < " + number2 );
30
31       if ( number1 > number2 )
32          Console.WriteLine( number1 + " > " +  number2 );
33
```

Comparison.cs

```
34       if ( number1 <= number2 )
35          Console.WriteLine( number1 + " <= " + number2 );
36
37       if ( number1 >= number2 )
38          Console.WriteLine( number1 + " >= " + number2 );
39
40    } // end method Main
41
42 } // end class Comparison
```

Program Output

```
Please enter first integer: 1000

Please enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Please enter first integer: 2000

Please enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

```
Please enter first integer: 1000

Please enter second integer: 1000
1000 == 1000
1000 <= 1000
1000 >= 1000
```