Programming

Join the discussion @ p2p.wrox.com
wrox
Wrox Programmer to Programmer™
Professional
C# 7 and
.NET Core 2.0
Christian Nagel

Lecture 3

Introduction to C# and .NET

Topics

- Introduction to C# and .NET
  - Features of C#
  - .NET Framework
  - C# basic syntax

➢ Basic usage of Visual Studio .NET
  - o Basic features
  - o Create a new project
  - o Compile & Run
  - o Basic of debugging

Introduction C#

- Introduced in 2000 by Microsoft
  - Has roots in the C, C++, and Java
- It’s appropriate for the most demanding app-development tasks
  - Large-scale enterprise
  - Web-based, mobile and “cloud”-based apps

Object-Oriented Programming

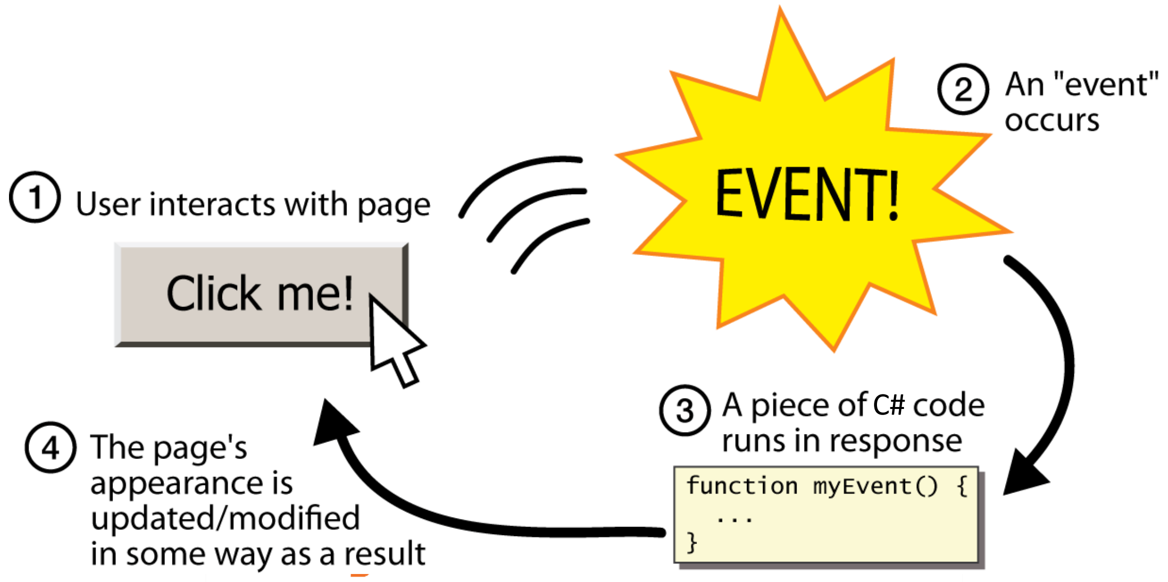
- C# is object oriented
- C# has access to the powerful .NET Framework Class Library
  - Vast collection of built-in classes to develop app quickly
  - We will learn more about .NET Framework later

Some key capabilities in the .NET Framework Class Library

Database	Debugging
Building web apps	Multithreading
Graphics	File processing
Input/output	Security
Computer networking	Web communication
Permissions	Graphical user interface
Mobile	Data structures
String processing	

Event-Driven Programming

- C# is event driven.
  - We write programs to respond to user-initiated events e.g., mouse clicks, keystrokes, timer expiration, etc
  - Or touches, finger swipes, etc on smartphones



1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of C# code runs in response
function myEvent() {
...
}
4 The page's appearance is updated/modified in some way as a result

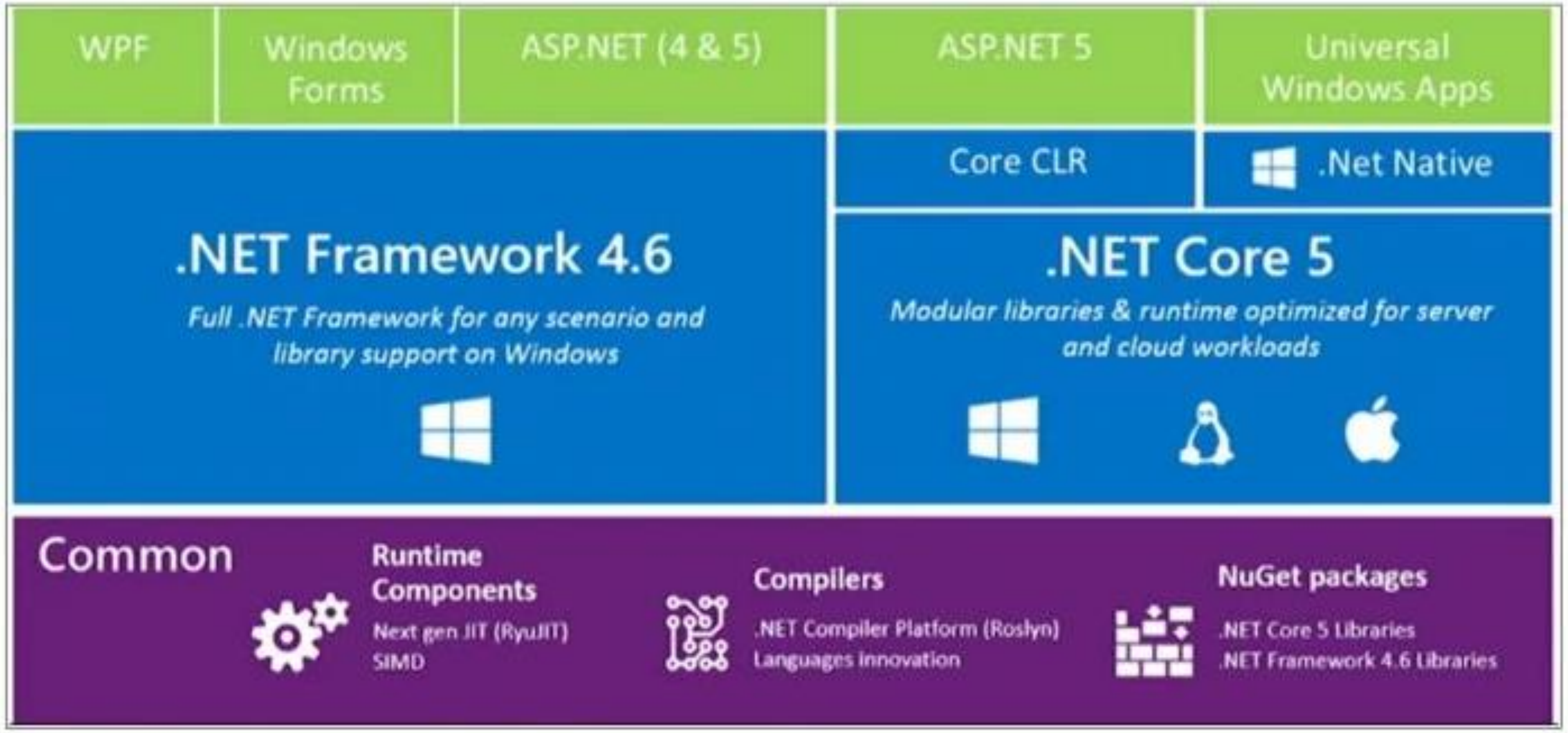
Visual Programming

- C# is visual programming language
  - You can write code
  - You can also use VS to drag/drop and design GUI
  - Then VS will write the GUI code for you
  - Allows us to focus on coding business processing

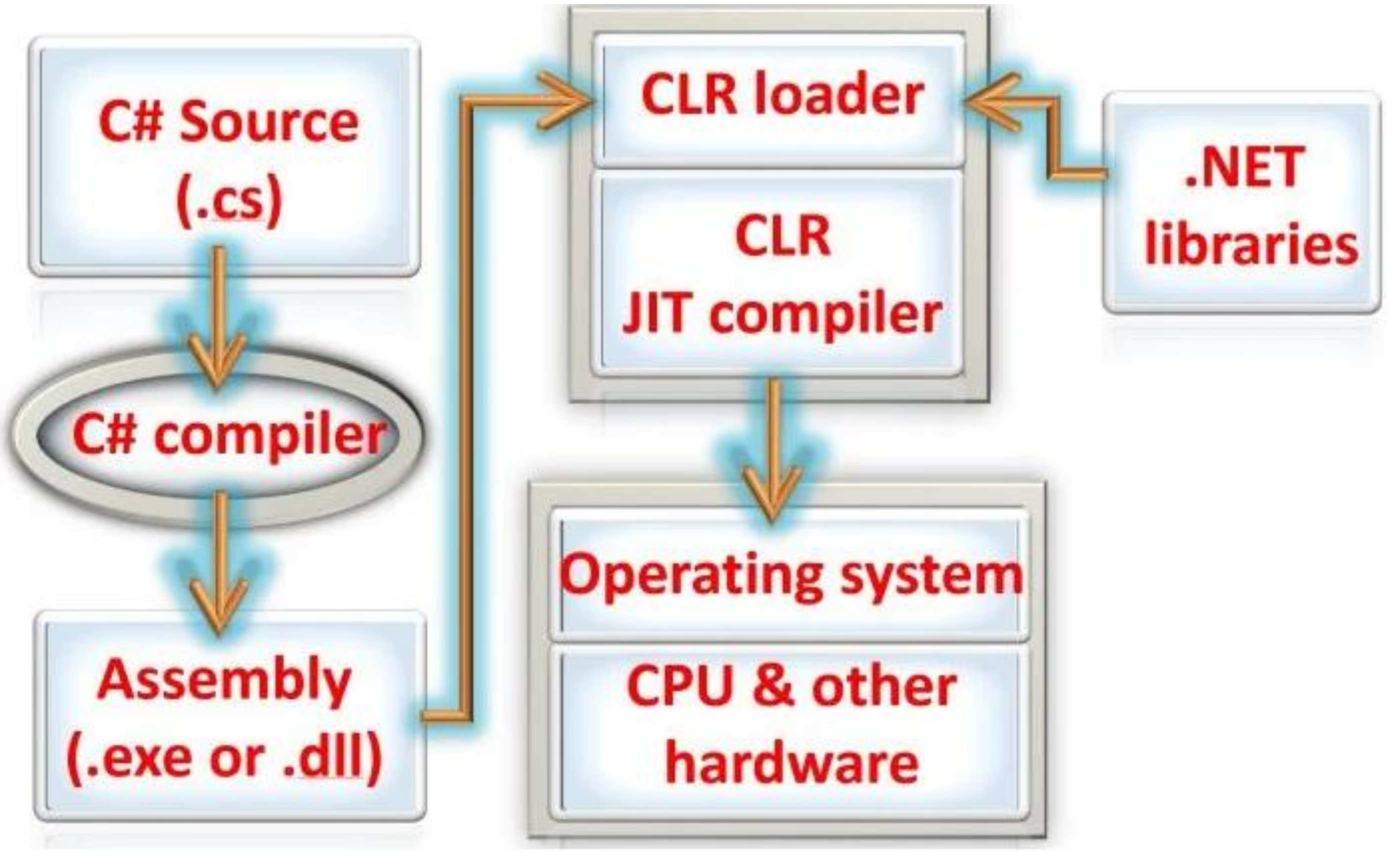
An International Standard

- C# has been standardized internationally
  - Enables other implementation of language besides MS’s Visual C#
  - One example is Mono that runs on Linux, iOS, Android, and Windows

How C# works


WPF
Windows Forms
ASP.NET (4 & 5)
ASP.NET 5
Universal Windows Apps
Core CLR
.Net Native
.NET Framework 4.6
Full .NET Framework for any scenario and library support on Windows
.NET Core 5
Modular libraries & runtime optimized for server and cloud workloads
Common
Runtime Components
Next gen JIT (RyuJIT)
SIMD
Compilers
.NET Compiler Platform (Roslyn)
Languages innovation
NuGet packages
.NET Core 5 Libraries
.NET Framework 4.6 Libraries

How C# works


C# Source (.cs)
C# compiler
Assembly (.exe or .dll)
CLR loader
CLR JIT compiler
.NET libraries
Operating system
CPU & other hardware

Internet and Web Programming

- We can build web based app with C# and Microsoft's ASP.NET technology

Microsoft’s .NET

Introduction

- In 2000 by Microsoft announce .NET initiative
  - Broad vision for using the Internet and the web in the development, engineering, distribution and use of software
  - .NET permits you to create apps using any .NET-compatible langue (C#, Visual Basic, Visual C++…)
  - Part of the initiative includes ASP.NET technology

.NET Framework

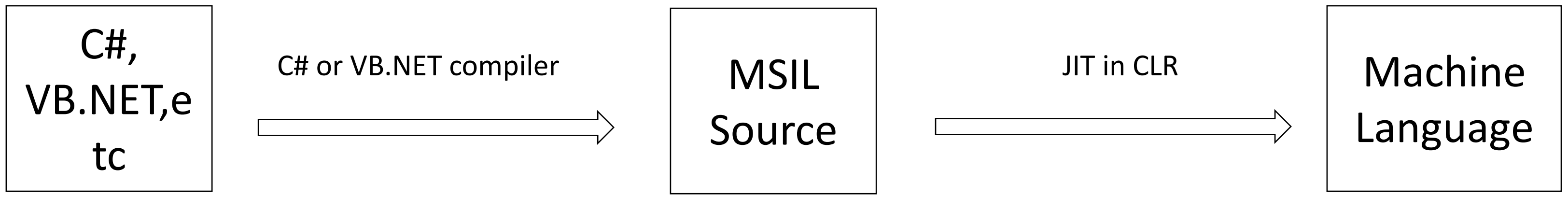
- It executes apps and contains the Class Library
- .NET Framework Class Library
  - Contains many valuable prebuilt classes
  - These classes are tested and tuned
  - These speedup the development & performance

Common Language Runtime (CLR)

- CLR is another key part of the .NET Framework
  - Executes .NET programs and provides functionality to make easier to develop and debug
- CLR is a virtual machine (VM)
  - It manages execution of programs and hides from them the underlying operating system hardware
  - Source code are executed/managed by CLR is called managed code.
- CLR provides many services to managed code
  - Integrating software components written in different .NET languages
  - Error handling between such components
  - Enhanced security
  - Automatic memory management
  - Etc.

Managed Code to Machine Instruction

- Managed code is compiled into machine-specific instructions in following steps
  1. First code is compiled into MSIL (all C#, J#, etc will be compiled into MSIL)
  2. When the app executes, JIT compiler in the CLR translates MSIL into machine code
  3. The machine code executes on that platform



Platform Independence

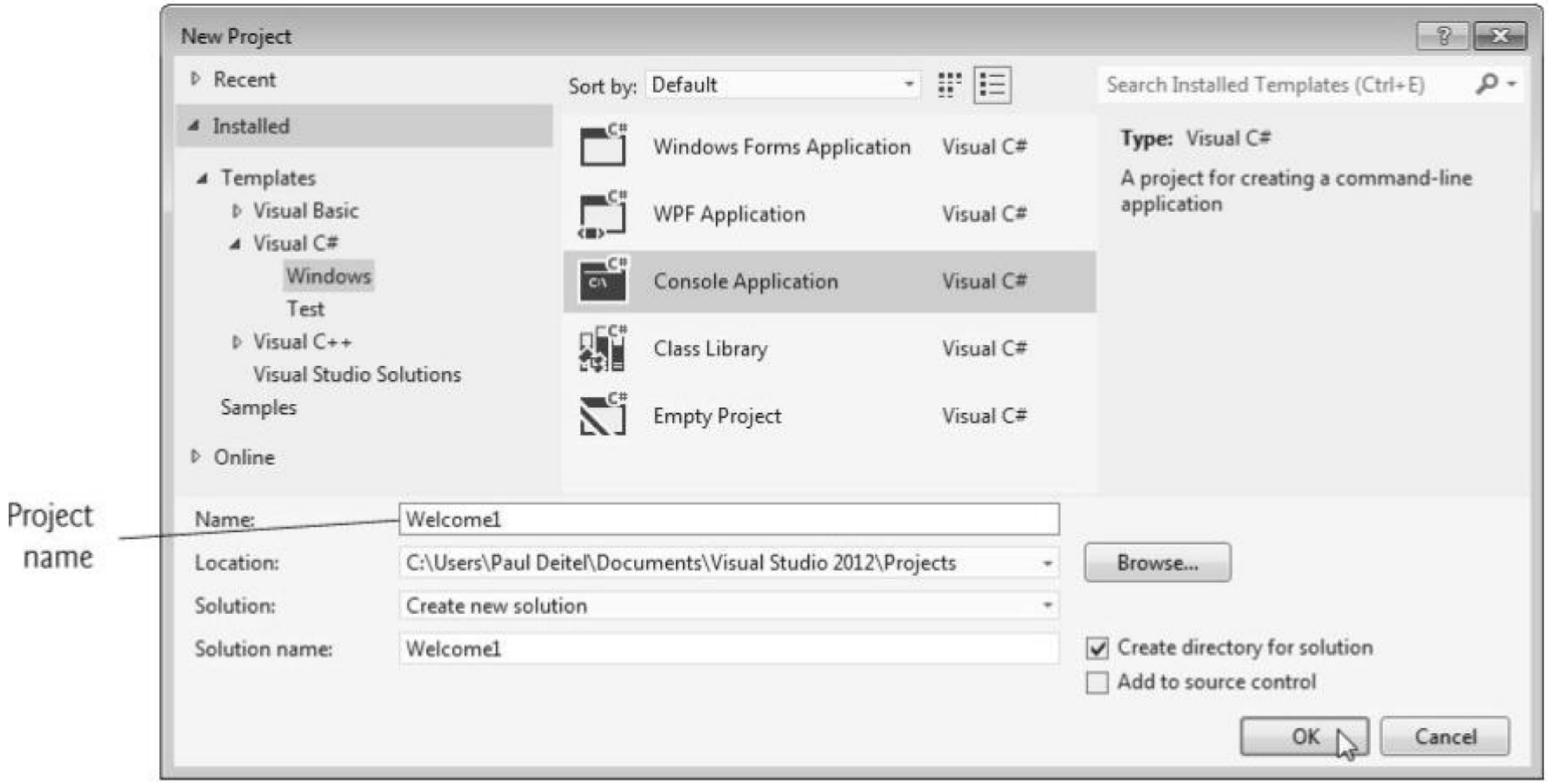
- .NET Framework exists and is installed for a platform
- .NET is platform independence
  - Ability to run across multiple platforms
  - E.g., we can install .NET in Mac OS

Language Interoperability

- .NET Framework provides high level language interoperability
  - Software can be written in C#, Visual Basic, etc
  - All will be compiled into MSIL

C# Basic Syntax

Creating a new app



New Project

- Recent
- Installed
  - Templates
    - Visual Basic
    - Visual C#
      - Windows
      - Test
    - Visual C++
    - Visual Studio Solutions
  - Samples
- Online

Sort by: Default

Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Class Library	Visual C#
Empty Project	Visual C#

Type: Visual C#

A project for creating a command-line application

Name: Welcome1

Location: C:\Users\Paul Deitel\Documents\Visual Studio 2012\Projects

Solution: Create new solution

Solution name: Welcome1

Browse...

[✓] Create directory for solution

[ ] Add to source control

OK Cancel

The source code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloWorld
{

//A Simple Program to display the words Hello World

class Program
{
static void Main(string[] args)
{
Console.WriteLine("Hello World");
Console.Read();
}
}
}
```

[illegible]

using Directive

- From line 1 to 5, we have a few statements that start with the word using. These statements are known as directives. They tell the compiler that our program uses a certain namespace
- For instance, the first line

```
using System;
```

- tells the compiler that our program uses the System namespace.

Namespace

- A namespace is simply a grouping of related code elements. These elements include classes, interfaces, enums and structs etc
- C# comes with a large amount of pre-written code that are organised into different namespaces.
- The System namespace contains code for methods that allow us to interact with our users. We use two of these methods in our program - the WriteLine() and Read() methods

Class declaration

- Every app consists of at least one class declaration that defined by the programmer
  - user-defined class
  - E.g., public class Welcome1
- Class Name convention (upper camel casing)
  - Begin with a capital letter
  - Capitalize the first letter of each word included
  - Contains letters, digits, and underscore
  - Doesn’t start with digit, doesn’t contain spaces
- C# is case sensitive
  - So be careful because Myname is different from MyName

C# Method and Statements

- Main method
  - public static void Main(string[] args)
  - It’s the starting point of every app
- Statements
  - Statements end with a semicolon (;)

statements/comments

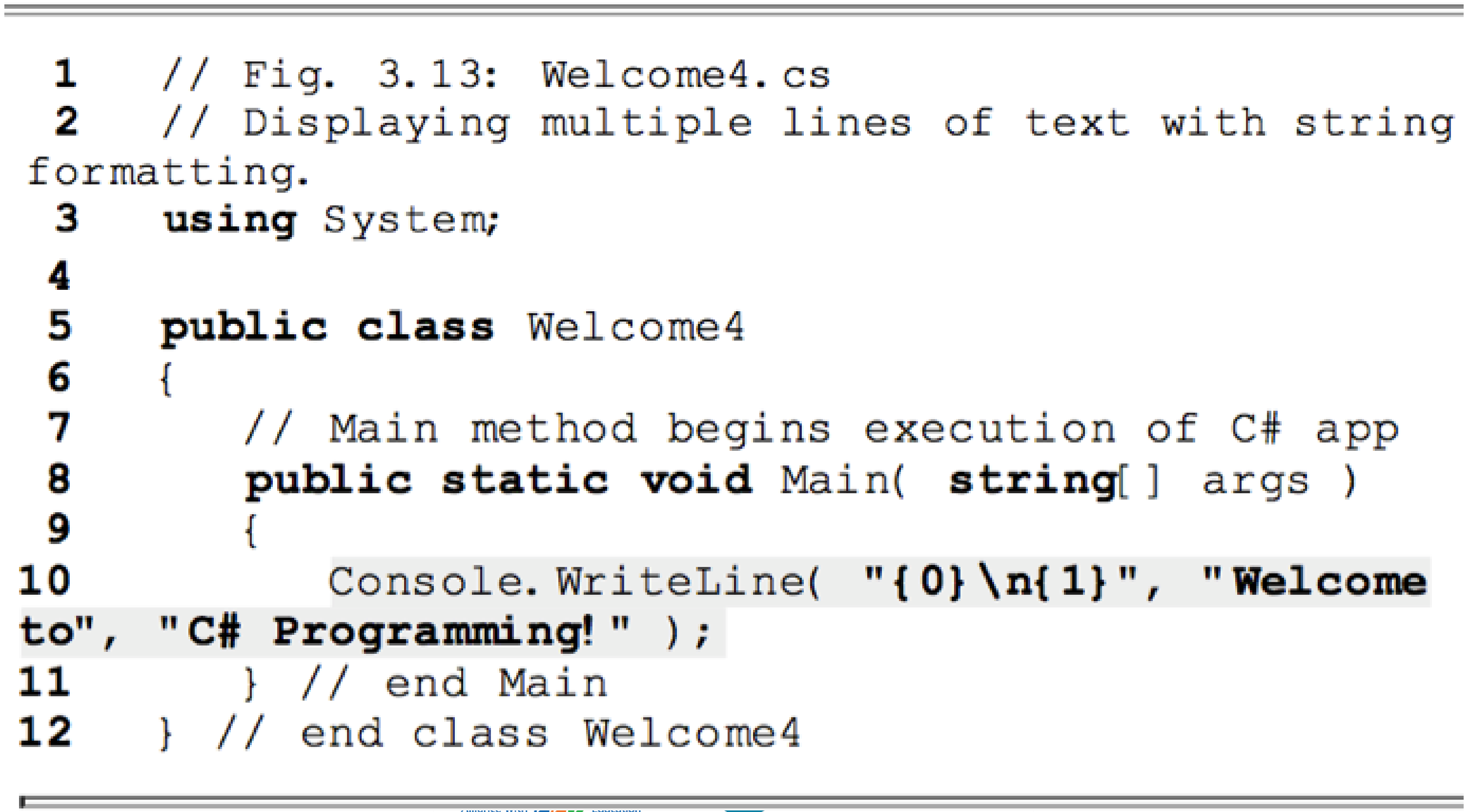
- Statement is instruction that programmer tells computer to do
  - Every C# statement is terminated with semicolon ‘;’
- Comment is explanation / guide that programmer explain to reader some block of code and comments will not be executed
  - Comment a single line with //
  - Comment a block with /*……….*/

Multiple Line Statement

```
// Fig. 3.10: Welcome2.cs
// Displaying one line of text with multiple statements.
using System;

public class Welcome2
{
   // Main method begins execution of C# app
   public static void Main( string[] args )
   {
      Console.Write( "Welcome to " );
      Console.WriteLine( "C# Programming!" );
   } // end Main
} // end class Welcome2
```

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

Formatting Text



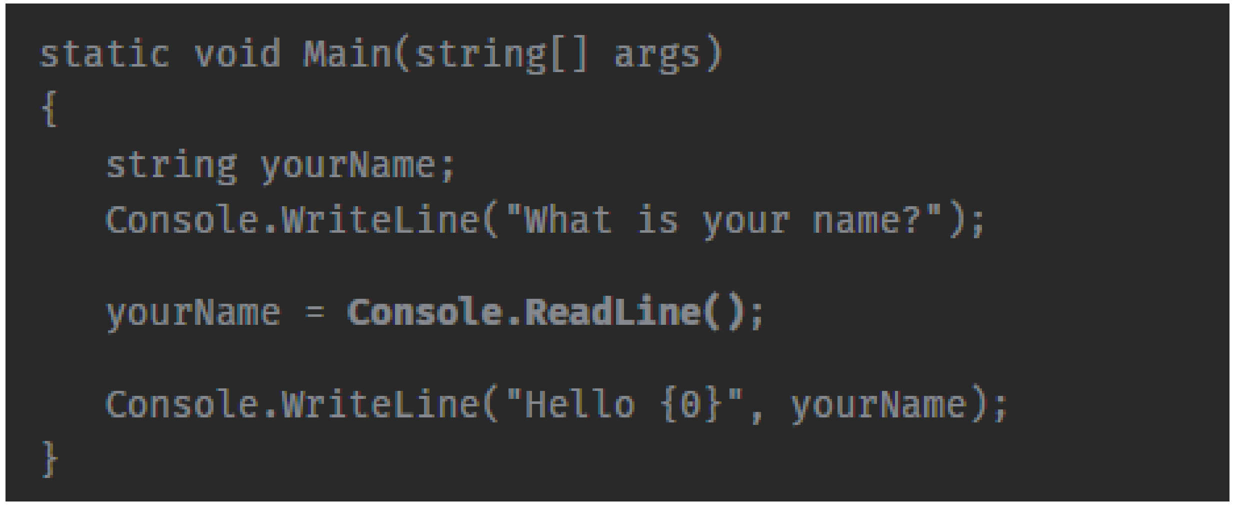
```
// Fig. 3.13: Welcome4.cs
// Displaying multiple lines of text with string formatting.
using System;

public class Welcome4
{
   // Main method begins execution of C# app
   public static void Main( string[] args )
   {
      Console.WriteLine( "{0}\n{1}", "Welcome to", "C# Programming!" );
   } // end Main
} // end class Welcome4
```

Displaying Output

- Most applications require some **input** from the user and give **output** as a result.
  To display text to the console window you use the **Console.Write** or **Console.WriteLine** methods.
- The difference between these two is that **Console.WriteLine** is followed by a line terminator, which moves the cursor to the next line after the text output.
- The program below will display Hello World! to the console window:

User Input

- You can also prompt the user to enter data and then use the **Console.ReadLine** method to assign the input to a string variable. The following example asks the user for a name and then displays a message that includes the input:



```
static void Main(string[] args)
{
   string yourName;
   Console.WriteLine("What is your name?");

   yourName = Console.ReadLine();

   Console.WriteLine("Hello {0}", yourName);
}
```

User Input

- The **Console.ReadLine()** method returns a **string** value.
  If you are expecting another type of value (such as int or double), the entered data must be converted to that type.

- This can be done using the **Convert.ToXXX** methods, where XXX is the .NET name of the type that we want to convert to. For example, methods include **Convert.ToDouble** and **Convert.ToBoolean**.
- For integer conversion, there are three alternatives available based on the bit size of the integer: **Convert.ToInt16**, **Convert.ToInt32** and **Convert.ToInt64**. The default int type in C# is 32-bit.

Read the input

- For standard input we use Console.ReadLine()
  - String s = Console.ReadLine();
  - int n = Convert.ToInt32(Console.ReadLine());
- Possible erroneous input
  - User can input a string which is not an integer
  - In this case an Exception is raised
  - We will learn about handling exception later