Chapter 1 and Introduction to Book Overview

Course Overview: contents of the book at a glance

**Source Code download**: <https://github.com/Apress/pro-csharp-7>

**Two goals of the book:**

1. teach syntax and semantics of C#
2. teach .NET APIs: ADO.NET, EF, WPF, WCF, desktop and web development

**Building blocks of the .NET platform**

platform and language independent run time execution engine and base class libraries

* MSIL (CIL)
* JIT
* CLR
* CTS
* CLS

figures 1-1, 1-2, 1-3

example on page 11 Calc.cs

review IL code with: ildasm.exe: C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools

review manifest and metadata on a DLL

multi language world of .net: c# .net, vb .net, f# .net, cobol.net

**History of .NET**

understanding CTS: class, interface, structure, enum, delegate

understanding intrinsic CTS data types: page 18

**Assembly/Namespace/Type**: pg 22 System.IO, System.Data keep names from conflicting

Sample of .NET namespaces

M$ root namespace – not safe for cross platform

using keyword

**GAC**: c:\windows\assembly\GAC

**Platform independent versions of .NET**

**Visual Studio Code**

Ch 2 Building C# Applications

Create a console application with VS – New Project Dialog

Running and Debugging

Solution Explorer

Adding references

Object Browser

Project Properties

Core C# Programming Constructs

j

**Console Paint Program**

Create a console paint program using Object Orientated Principles of encapsulation, inheritance, and polymorphism:

You will utilize the following class hierarchy:

A screenshot of a cell phone

Description generated with very high confidence

Shape class:

abstract base class for the Rectangle, Circle and Line class.

IBrush property – contains the brush to use to draw the shape.

Draw method - abstract method and therefore must be defined by all base classes

SetColor method - virtual protected method used to set the console color before the shape draws

ResetColor method - virtual protected method used to reset the console color back to the default after the shape draws. The default console color is ConsoleColor.Gray.

Rectangle, Circle, and Line classes extend Shape provide the draw method and prompt the user for any necessary variables.

IBrush interface:

Color – the console color that should be used for the drawing

Stroke – is the Unicode character that will be used to print the shape e.g. \_ or Ω, or Δ. A good resource to find Unicode characters and there numbers is here: <https://www.rapidtables.com/code/text/unicode-characters.html>

Resources

* for ascii shapes is here: shapes from here: <http://ascii.co.uk/art>
* Unicode <https://www.rapidtables.com/code/text/unicode-characters.html>

Example run:

c

Your main program should prompt for a shape and a brush. With that information you can create the appropriate Shape and Brush objects. Then you can call Draw on your shape.

Notice that each shape prompts for different parameters since they need different variables to draw. In the example above the Rectangle shape prompts for length and height whereas the circle shape prompts for a radius.

Chapter 9 Collections and Generics

Start with System.Collections namespace pg. 327

interfaces those collections implement pg. 328: ICollection, ICloneable, IDictionary, IEnumerable, IEnumerator, IList

generic collections are preferred because of type safety and better performance. Also generics prevent you from having to create your own type safe collection classes. pg. 337

generic placeholder <T> List<T> List of T pg. 338

System.Collections.Generic namespace: pg 342

Generic collection initialization syntax: pg. 344

FunWithGenericCollections pg. 345

FunWithObservableCollections pg. 350

GenericPoint pg. 357

Constraining Type Parameters pg. 360

**MyGeneric class.**

**Takes a generic of type G**

**create one property of type G called**

**GenericProperty**

**create one method on the class called**

**PrintGeneric()**

**that method will print the type and the value of GenericProperty**

Ch. 6 Notes

Understanding Inheritance and polymorphism

inheritance – promotes code reuse

1 base class

sealed keyword

another reason to choose classes over structures is c# structures are implicitly sealed

create a hierarchy of employees

call the base class ctor to set the base class properties

protected member variables in base classes

adding a sealed class

inner classes

virtual and override keywords

marking method abstract – Shape

using the as keyword to avoid InvalidCastException

is keyword to check for class type

Project: ASCII Art Store

concepts:

1. Given a problem statement provide a good object orientated design with
   1. classes needed
   2. events needed

problem statement:

ASCII Art store. Sell ASCII art. Provide an ASCII art store front in a console application.

Store

Inventory

ASCII Art

Admin functions

Sell Art

Manage Inventory

categories of ASCII art: animals, shapes, characters

Sale Specials

ArtRepository

Ch 11 Extension Methods

Add new methods or properties to a class or structure without adding them to the class directly or using inheritance

Check out the ExtensionMethods project under the books source folder.

You must import the namespace of extension methods to use them! Hint: Use using.

Intellisense help to indicate methods that are extensions.

Anonymous Types:

Allow for quick creation of data types to encapsulated properties for one-time use.

Extend from object

AnonymousTypes example project.

Ch. 12 Linq to objects

constructs needed to support LINQ – building blocks

* implicitly typed variables
* lambdas
* extensions methods
* anonymous types

Role of LINQ:



3 approaches (well really 2):

* + - 1. query expression – power tool with a laser light
      2. extension methods – power tool
      3. without linq (not really an approach to using linq) – hand tool

Deferred execution: I give you instructions on where to find what you need

Immediate execution: I give you what you need

101 LINQ Samples: <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>

Ch 11

**Call a method async using delegates**

Create a delegate

Invoke BeginInvoke on that method and store the IAsyncResult (calling BeginInvoke initializes and starts the background thread)

Invoke the EndInvoke method on the delegate (calling EndInvoke will wait for the delegate to complete and return the result if any)

Start with **AsyncDelegate**

starting a new thread with no callback

Then show **AsyncCallbackDelegate**

now invoke BeginInvoke with a callback

**Manually creating new threads**

ThreadStart and ParameterizedThreadStart

lock blocks of code

TimerCallback

**async await keywords**