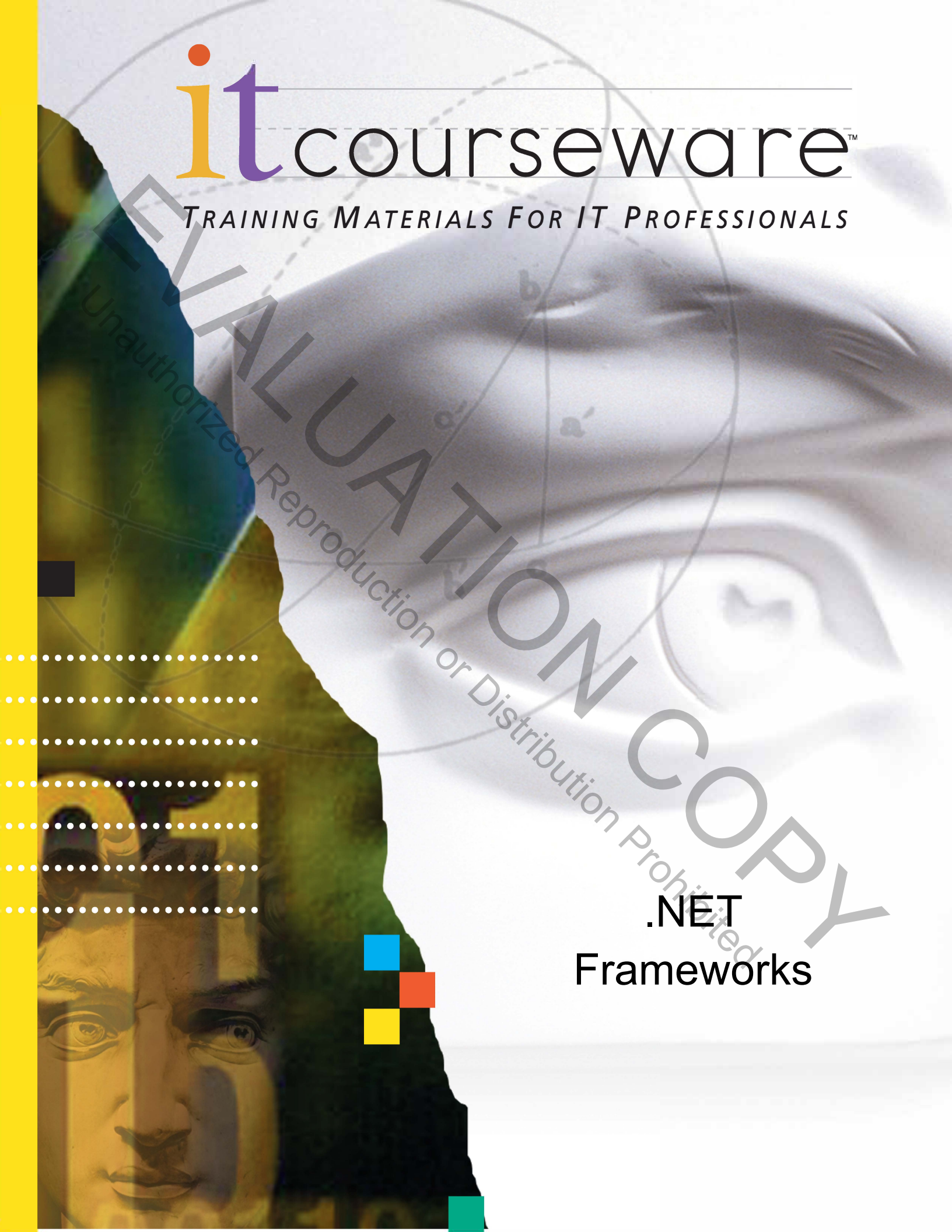
itcourseware™
TRAINING MATERIALS FOR IT PROFESSIONALS
EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited
.NET
Frameworks

.NET Frameworks
Rev. 5.0

Student Guide

™ is a trademark of Object Innovations.

Author: Robert J. Oberg

Object Innovations
877-558-7246
www.objectinnovations.net

Published in the United States of America.

Table of Contents (Overview)

Chapter 1	.NET Fundamentals
Chapter 2	Class Libraries
Chapter 3	Frameworks and Packages
Chapter 4	Metadata and Reflection
Chapter 5	I/O and Serialization
Chapter 6	Delegates and Events
Chapter 7	.NET Programming Model
Chapter 8	.NET Threading
Appendix A	Learning Resources

Directory Structure

- **The course software installs to the root directory *C:\OIC\NetCs*.**
  - Example programs for each chapter are in named subdirectories of chapter directories **Chap01**, **Chap02**, and so on.
  - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
  - The **Demos** directory is provided for hand-on work during lectures.
  - The **Packages** directory is for custom packages.
  - The **Run** directory is for .NET Core executables to be launched programmatically as a process.
  - The file **Links.htm** contains link to web sites and pages mentioned in the text along with other useful links.

Table of Contents (Detailed)

Chapter 1 .NET Fundamentals 1
What Is Microsoft .NET? 3
Open Standards and Interoperability 4
Windows Development Problems 5
Common Language Runtime 6
Serialization Example 7
Framework Programming 10
Types 11
.NET Framework Class Library 12
Everything is an Object 13
Common Type System 14
Language Interoperability 15
Managed Code 16
JIT Compilation 17
The Role of XML 18
Performance 19
.NET Core and .NET 5 20
Summary 21

Chapter 2 Class Libraries 23
Objects and Components 25
Limitation of COM Components 26
Components in .NET 27
Component Example: Customer Management System 28
Monolithic versus Component 30
Class Libraries Using Visual Studio 31
Demo: Creating a Class Library 32
Creating a Client Project 35
References in Visual Studio 37
References at Compile Time and Run Time 39
Project Dependencies 40
Lab 2 41
Summary 42

Chapter 3 Frameworks and Packages 47
.NET Implementations 49
Target Frameworks 50
.NET Standard 51
.NET Core and .NET Framework 52
Targeting .NET Framework 53
Targeting .NET Core 54
A .NET Core Console App 55

What is NuGet? ... 56
Packages, Metapackages and Frameworks ... 57
Packages ... 58
Key Packages for .NET Core ... 59
Metapackages ... 60
Frameworks ... 61
.NET Core API Reference ... 62
.NET API Browser ... 63
Lab 3A ... 64
Example: File Output ... 65
Demo: A Windows Forms App ... 66
Dependencies ... 72
NuGet Package Manager ... 73
Installed Packages ... 77
Creating a Package from a Class Library ... 79
Example: CustomerLib Package ... 80
Putting a Package in a NuGet Source ... 83
Individual NuGet Accounts ... 84
NuGet Home Page (Signed In) ... 86
Publish To nuget.org ... 87
Using Your Package ... 93
Lab 3B ... 94
Summary ... 95

Chapter 4 Metadata and Reflection ... 103
Metadata ... 105
Reflection ... 106
Sample Reflection Program ... 107
System.Reflection.Assembly ... 110
System.Type ... 111
System.Reflection.MethodInfo ... 113
Dynamic Invocation ... 114
Late Binding ... 115
LateBinding Example ... 116
Lab 4 ... 117
Summary ... 118

Chapter 5 I/O and Serialization ... 123
Input and Output in .NET ... 125
Directories ... 126
Directory Example Program ... 127
Files and Streams ... 130
“Read” Command ... 132
Code for “Write” Command ... 133
Serialization ... 134
XML Serialization Demo ... 135

Code Walkthrough 136
XML Serialization Infrastructure 141
What Will Not Be Serialized 142
Lab 5 143
Summary 144

Chapter 6 Delegates and Events 149
Overview of Delegates and Events 151
Callbacks and Delegates 152
Usage of Delegates 153
Declaring a Delegate 154
Defining a Method 155
Creating a Delegate Object 156
Calling a Delegate 157
Random Number Generation 158
A Random Array 159
Combining Delegate Objects 160
Account.cs 161
Program.cs 162
Running the Example 163
Anonymous Methods 164
Lambda Expressions 165
Named Method Example 166
Anonymous Method Example 167
Lambda Expression Example 168
Events 169
Events in C# and .NET 170
Client Side Event Code 172
Chat Room Example 173
Lab 6 174
Summary 175

Chapter 7 .NET Programming Model 179
Garbage Collection 181
Finalize Method 182
C# Destructor Notation 183
Dispose 184
Garbage Collection Performance 185
Generations 186
Processes 187
Application Isolation 188
Process Example 189
Properties, Methods and Events 191
Exited Event 192
Lab 7 193
Command Line Arguments 194

SimpleLog Class 195
Arguments in Visual Studio 196
Starting a .NET Core Process 197
The Run Directory 199
Process Launch in .NET 5.0 200
Threads 201
Summary 202

Chapter 8 .NET Threading 205
Threads 207
.NET Threading Model 208
ThreadDemo Example 209
Sample Output 213
Race Conditions 214
Race Condition Example 215
Race Condition Output 219
Thread Synchronization 220
Monitor 221
Monitor Example 222
Monitor Example Output 223
Using C# *lock* Keyword 224
Synchronization of Collections 225
ThreadPool Class 226
ThreadPool Example 227
Starting a ThreadPool Thread 228
Foreground and Background Threads 229
Synchronizing Threads 230
Improved ThreadPool Example 231
Task Parallel Library (TPL) 233
Task Example 234
Starting Tasks 235
Waiting for Task Completion 236
Data Parallelism 237
Lab 8 238
Summary 239

Appendix A Learning Resources 247

Chapter 1

.NET Fundamentals

.NET Fundamentals

Objectives

After completing this unit you will be able to:

- **Understand the problems Microsoft .NET is designed to solve.**
- **Understand the basic programming model of Microsoft .NET.**
- **Understand the basic programming tools provided by Microsoft .NET.**
- **Discuss .NET Core, .NET 5 and cross-platform development.**

What Is Microsoft .NET?

- **This chapter discusses fundamental concepts of .NET in general.**
  - The specifics of .NET Core will be covered in the rest of the course.
  - The programming example in this chapter does use .NET Core.
- **Microsoft .NET was developed to solve three fundamental problems.**
  - First, the Microsoft Windows programming model must be unified to remove the widely varied programming models and approaches that exist among the various Microsoft development technologies.
  - Second, Microsoft based solutions must be capable of interacting with the modern world of heterogeneous computing environments.
  - Third, Microsoft needs a development paradigm that is capable of being expanded to encompass future development strategies, technologies, and customer demands.

Open Standards and Interoperability

- **The modern computing environment contains a vast variety of hardware and software systems.**
  - Computers range from mainframes and high-end servers, to workstations and PCs, and to small mobile devices such as PDAs and smartphones.
  - Operating systems include traditional mainframe systems, many flavors of Unix including Android, Linux, Apple's iOS, several versions of Windows, real-time systems and more.
  - Many different languages, databases, application development tools and middleware products are used.
- **Applications need to be able to work in this heterogeneous environment.**
  - Even shrink-wrapped applications deployed on a single PC may use the Internet for registration and updates.
- **The key to interoperability among applications is the use of *standards*, such as HTML, HTTP, XML, and TCP/IP.**

Windows Development Problems

- **In classic Windows development design and language choice often clashed.**
  - Visual Basic vs. C++ approach
  - IDispatch, Dual, or Vtable interfaces
  - VB vs. MFC
  - ODBC or OLEDB or ADO
- **Application deployment was hard.**
  - Critical entries in Registry for COM components
  - No versioning strategy
  - DLL Hell
- **Too much time is spent in writing plumbing code that the system should provide.**
  - MTS/COM+ was a step in the right direction.

Common Language Runtime

- **The first step in solving the three fundamental problems is for Microsoft .NET to provide a set of underlying services available to all languages.**
- **The runtime environment provided by .NET that provides these services is called the *Common Language Runtime* or CLR.**
  - A runtime provides services to executing programs.
  - Traditionally there are different runtimes for different programming environments. Examples of runtimes include the standard C library, MFC, the Visual Basic 6 runtime and the Java Virtual Machine.
- **These services are available to all languages that follow the rules of the CLR.**
  - .NET languages include C#, F# and Visual Basic.
  - Object Innovations' curriculum emphasizes C#.

Serialization Example

- **Let us use serialization to illustrate how the CLR provides a set of services that unifies the Microsoft development paradigm.**
  - Every programmer has to do it.
  - It can get complicated with nested objects, complicated data structures, and a variety of data storages.
- **However, serialization can open up security vulnerabilities.**
  - In particular, Binary Serialization, part of the classical .NET Framework, is insecure and cannot be made secure. It should be avoided.
  - See this article for more information:

https://docs.microsoft.com/en-us/dotnet/standard/serialization/binaryformatter-security-guide

- **The *Serialize* example in this chapter uses XmlSerialization.**

Serialization Example (Cont'd)

- **Ignore the language details covered in a later chapter.**

```
class Customer
{
   public string name;
   public long id;
}
class Program
{
   static void Main(string[] args)
   {
      List<Customer> list = new List<Customer>();

      Customer cust = new Customer();
      cust.name = "Charles Darwin";
      cust.id = 10;
      list.Add(cust);

      cust = new Customer();
      cust.name = "Isaac Newton";
      cust.id = 20;
      list.Add(cust);

      foreach (Customer x in list)
         Console.WriteLine(x.name + ": " + x.id);

      Console.WriteLine("Saving Customer List");

      XmlSerializer ser = new
         XmlSerializer(typeof(List<Customer>));
      FileStream s = new FileStream("cust.xml",
                                    FileMode.Create);
      ser.Serialize(s, list);
      s.Close();
      ...
```

Serialization Example (Cont'd)

```
        Console.WriteLine("Restoring to New List");

        FileStream s2 =
           new FileStream("cust.xml", FileMode.Open);
        list = (List<Customer>)ser.Deserialize(s2);

        foreach (Customer y in list2)
           Console.WriteLine(y.name + ": " + y.id);
        s2.Close();
    }
```

Framework Programming

- **We add two *Customer* objects to the collection, and print them out. We save the collection to disk and then restore it. The identical list is printed out.**

```
Charles Darwin: 10
Isaac Newton: 20
Saving Customer List
Restoring to New List
Charles Darwin: 10
Isaac Newton: 20
Press enter to continue...
```

- **We wrote little code to save or restore the list!**
  - Nothing special was needed in the **Customer** class.
  - We used several .NET Framework classes to do the heavy lifting for us in the **Main()** method.
- **Classes are used throughout .NET to provide useful functionality, such as manage collections, do I/O and work with XML.**
  - The data file created was **cust.xml**.

Types

- ***Types* are at the heart of the programming model for the CLR[1].**
- **A type is analogous to a class in most object-oriented programming languages, providing an abstraction of data and behavior, grouped together.**
- **A type in the CLR contains:**
  - Fields (data members)
  - Methods
  - Properties
  - Events (which are now full fledged members of the Microsoft programming paradigm).

[1] We are using "CLR" in a generic sense. It is CLR for the classical .NET Framework and CoreCLR for .NET Core.

.NET Framework Class Library

- **The *List<T>*, *XMLFormatter* and *FileStream* classes are some of the thousands of classes in the .NET Framework that provide system services.**
- **The functionality provided includes:**
  - Base Class Library (basic functionality such as strings, arrays and formatting).
  - Networking
  - Security
  - Diagnostics
  - I/O
  - Database
  - XML
  - Web programming
- **This framework is usable by all CLR compliant languages.**

Everything is an Object

- **Every type in .NET derives from *System.Object*.**[2]
- **Every type, system or user defined, has metadata.**
  - In the sample the framework can walk through the List of Customer objects and save each one as well as the array itself.
- **All access to objects in .NET is through object references.**

[2] An exception is the pointer type, which is rarely used in C#.

Common Type System

- **The Common Type System (CTS) defines the rules for the types and operations that the CLR will support.**
  - The CTS limits .NET classes to single implementation inheritance.
  - The CTS is designed for a wide range of languages, not all languages will support all features of the CTS.
- **The CTS makes it *possible* to guarantee type safety.**
  - Access to objects can be restricted to object references (no pointers), each reference refers to a defined memory. Access to that layout is only through public methods and fields.
  - By performing a local analysis of the class, you can verify to make sure that the code does not perform any inappropriate memory access. You do not have to analyze the users of the class.
- **.NET compilers emit Microsoft Intermediate Language (MSIL or IL) not native code.**
  - MSIL is platform independent.
  - Type-safe code can be restricted to a subset of verifiable MSIL expressions.
  - Once code is verified, it is verified for all platforms.

Language Interoperability

- **Having all language compilers use a common intermediate language and common base class makes it *possible* for languages to interoperate.**
  - All languages need not implement all parts of the CTS.
  - One language can have a feature that another does not.
- **The *Common Language Specification* (CLS) defines a subset of the CTS that represents the basic functionality that all .NET languages should implement if they are to interoperate with each other.**
  - For example, a class written in Visual Basic can inherit from a class written in C#.
  - Interlanguage debugging is possible.
  - CLS rule: Method calls need not support a variable number of arguments even though such a construct can be expressed in MSIL.
  - CLS prohibits the use of pointers.
  - CLS is an ECMA specification.
- **CLS compliance only applies to public features.**
  - C# code should not define public and protected class names that differ only by case sensitivity since languages such as Visual Basic are not case sensitive. Private C# fields could have such names.

Managed Code

- **In the serialization example we never freed any allocated memory.**
  - Memory that is no longer referenced can be reclaimed by the CLR's garbage collector.
  - Automatic memory management eliminates the common programming error of memory leaks.
  - Garbage collection is one of the services provided to .NET applications by the Common Language Runtime.
- **Managed code uses the services of the CLR.**
  - MSIL can express access to unmanaged data in legacy code.

JIT Compilation

- **Before executing on the target machine, MSIL is translated by a just-in-time (JIT) compiler to native code.**
- **Some code typically will never be executed during a program run.**
  - Hence it may be more efficient to translate MSIL as needed during execution, storing the native code for reuse.
- **When a type is loaded, the loader attaches a stub to each method of the type.**
  - On the first call the stub passes control to the JIT, which translates to native code and modifies the stub to save the address of the translated native code.
  - On subsequent calls to the method transfer is then made directly to the native code.
- **As part of JIT compilation code goes through a verification process.**
  - Type safety is verified, using both the MSIL and metadata.
  - Security restrictions are checked.

The Role of XML

- **XML is ubiquitous in .NET and is highly important in Microsoft's overall vision.**
- **Some uses of XML in .NET include:**
  - XML can be used for serialization.
  - XML is used extensively in configuration files.
  - XML documentation can be automatically generated by .NET languages.
  - .NET classes provide a very convenient API for XML programming as an alternative to DOM or SAX.

Performance

- **Concerns about performance of managed code are similar to the concerns assembly language programmers had with high level languages.**
- **Garbage collection usually produces faster allocation than C++ unmanaged heap allocation. Deallocation is done on a separate thread by the garbage collector.**
- **JIT compilation takes a hit the first time when verification and translation take place, but subsequent executions pay no penalty.**
- **Bottom line: for most of the code that is written, any small loss in performance is far outweighed by the gains in reliability and ease of development.**
  - High performance servers might still have to use technologies such as C++.
- **Apps targeting the Windows 10 platform may use .NET Native to achieve higher performance.**

.NET Core and .NET 5

- **.NET Core is a modular version of the .NET Framework that is portable across multiple platforms.**
  - .NET Core represents the future of .NET.
  - Latest version of .NET Core is .NET 5.0.
- **Rather than one large assembly, .NET Core is released through NuGet in smaller feature-specific assembly packages.**
  - The *metapackage* convention allows a set of packages that are meaningful together to be treated as a unit.
- **.NET Core provides key functionality used in applications regardless of platform.**
  - This common functionality provides for shared code that can be used across platforms.
  - Your application then links in additional platform-specific code.
- **Microsoft platforms you can target include traditional desktop Windows and Windows phones.**
- **Other platforms include Mac and Linux.**
- **With Xamarin (now owned by Microsoft) you can target Android and iOS mobile platforms.**

Summary

- **.NET solves problems of past Windows development.**
- **One development paradigm for all languages exists.**
- **Design and programming language no longer conflict.**
- **.NET uses managed code with services provided by the Common Language Runtime that uses the Common Type System.**
- **Plumbing code for fundamental system services is there, yet you can extend it or replace it if necessary.**
- **The .NET Framework is a very large class library available consistently across many languages.**
- **.NET Core is a modular version of the .NET Framework that is portable across multiple platforms.**
  - The latest version of .NET Core is .NET 5.0.

7400 E. Orchard Road, Suite 1450 N
Greenwood Village, Colorado 80111
Ph: 303-302-5280
www.ITCourseware.com