INFORMATION PROCESSING TECHNIQUES

Week 01

Murtaza Munawar Fazal

CONSULTATION HOURS

- Available only on class day(s)
  - As per timetable version 1.0:
    - Tuesday : 10:00 am – 2:00 pm
    - Thursday : 10:00 am – 2:00 pm

- For any queries after the consultation hours, please don't hesitate to send me an email : murtaza.fazal@nu.edu.pk

CLASS RULES

- Attendance will be taken at the start of the class. Anyone coming after the attendance would be marked as Late. No leniency in attendance.
- Quizzes will be unannounced with no retakes.
- After every quiz/assignment, marks will be uploaded on google classroom. Make sure you verify all your marks.
- Keep your cell phones on silent.

ASSIGNMENT POLICY

- Assignment # 1:
  - If plagiarism is found, then 0 for that question
- Assignment # 2 and onwards:
  - If plagiarism is found, then 0 for entire assignment.

BOOKS

- *Reference Books*
  - *Pro C# 7 With .Net and .Net Core by Andrew Troelsen and Philip Japikse (Apress Publications)*

PRE-REQUISITE

- *You should have sound knowledge of programming*
- *You should have sound knowledge of data structures*
- *You should have sound knowledge of database server and writing queries*

TOOLS REQUIRED

- Visual Studio (VS 2017/2019/Code)
  - Windows Application
  - Windows Service
  - Web Application (ASP.Net)
  - Web Service (ASMX)
  - WCF
  - WEB API
- SQL Server (SQL Server 2014+ Professional or Enterprise Edition)

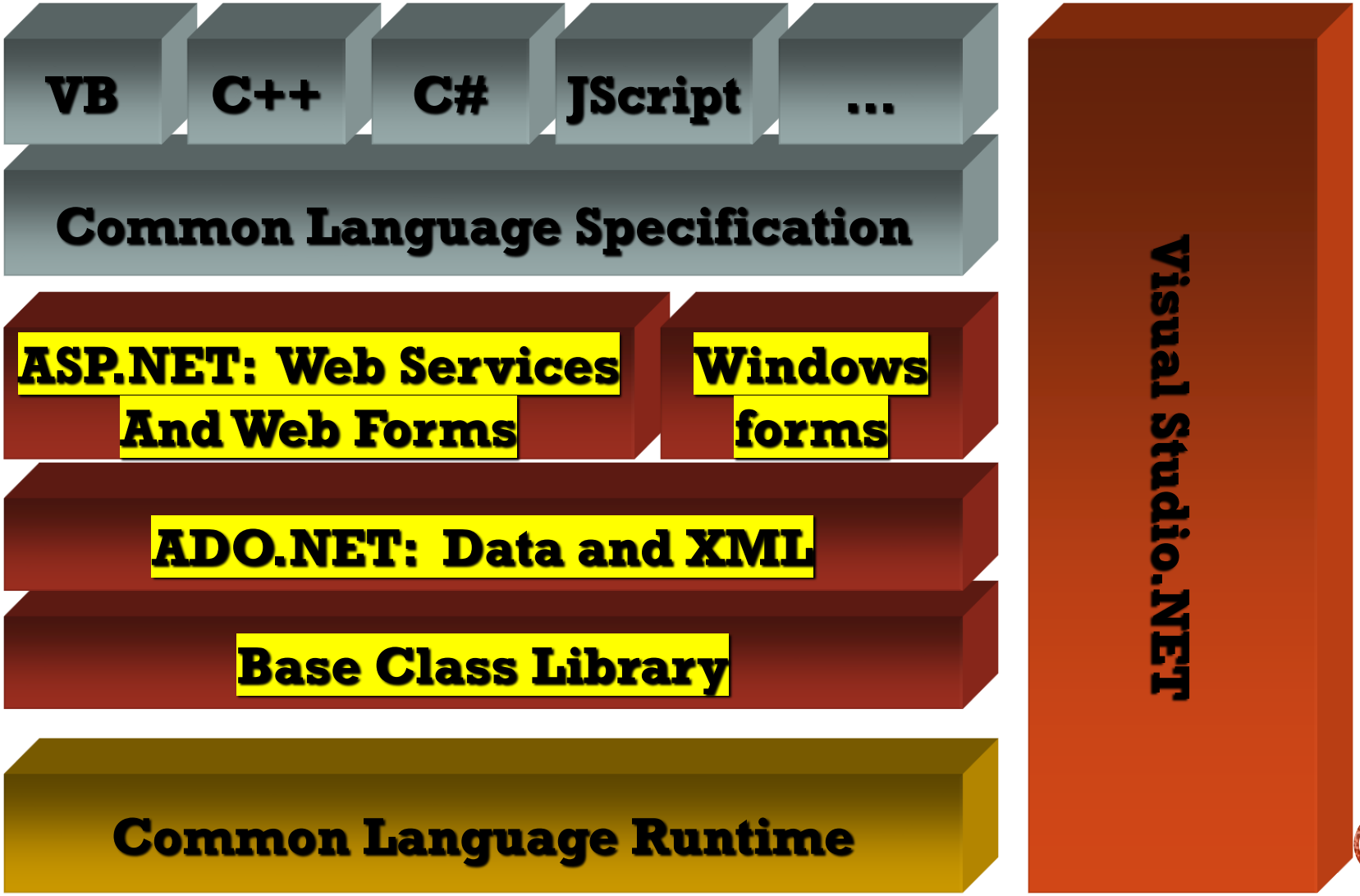
WHAT IS INFORMATION PROCESSING TECHNIQUE?

- Using technology efficient based on the scenario under consideration.

- Example:
  - Choosing Sorting Algorithm
  - Choosing Data Base , choosing types of join
  - Choosing different Web Service v/s WCF v/s Web API

THE .NET FRAMEWORK

OVERVIEW



COMMON LANGUAGE RUNTIME (CLR)

- The primary role of the CLR is to locate, load, and manage .NET objects.
- The CLR also takes care of a number of low-level details such as memory management, application hosting, coordinating threads, and performing basic security checks (among other low-level details).
- Common Classes for all Languages
- Common Types for all Languages
- Runtime Controls Compilation to Machine Code
- Assemblies

COMMON TYPE SYSTEM (CTS)

- The CTS specification fully describes all possible data types and all programming constructs supported by the runtime, specifies how these entities can interact with each other, and details how they are represented in the .NET metadata format

CTS Data Type	VB Keyword	C# Keyword	C++/CLI Keyword
System.Byte	Byte	byte	unsigned char
System.SByte	SByte	sbyte	signed char
System.Int16	Short	short	short
System.Int32	Integer	int	int or long
System.Int64	Long	long	__int64
System.UInt16	UShort	ushort	unsigned short
System.UInt32	UInteger	uint	unsigned int or unsigned long
System.UInt64	ULong	ulong	unsigned __int64
System.Single	Single	float	float
System.Double	Double	double	double
System.Object	Object	object	object^
System.Char	Char	char	wchar_t
System.String	String	string	String^
System.Decimal	Decimal	decimal	Decimal
System.Boolean	Boolean	bool	bool

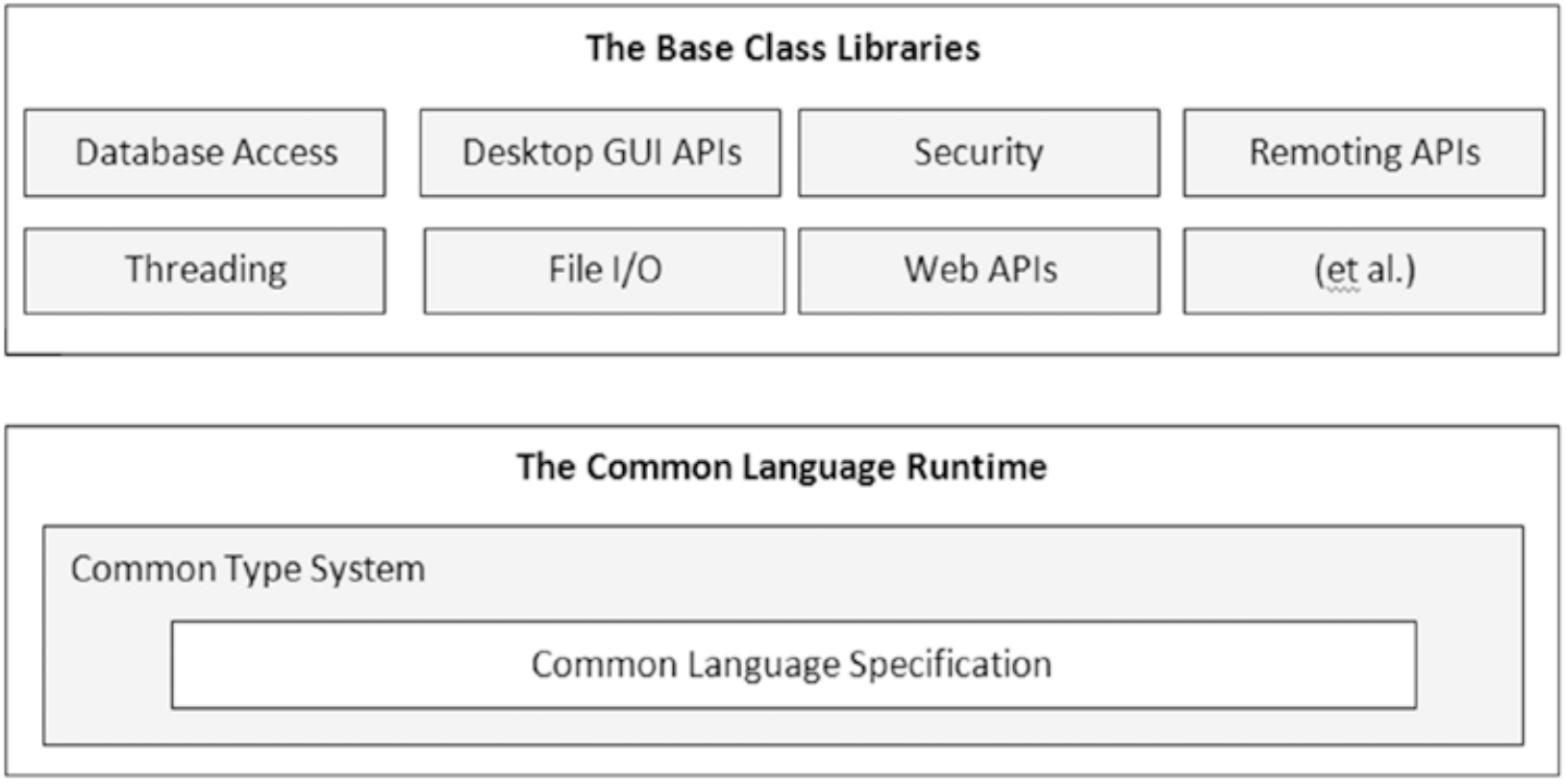
COMMON LANGUAGE SPECIFICATION (CLS)

- CLS is a related specification that defines a subset of common types and programming constructs that all .NET programming languages can agree on.
- if you build .NET types that expose only CLS-compliant features, you can rest assured that all .NET-aware languages can consume them.

BASE CLASS LIBRARY

- Available for all .Net languages.
- Encapsulate various primitives such as threads, file input/output (I/O), graphical rendering systems, and interaction with various external hardware devices
- It also provides support for a number of services required by most real-world applications.

THE CLR, CTS, CLS, AND BASE CLASS LIBRARY RELATIONSHIP



IMPORTANT POINTS…

- Pointers ?
- Memory Management (Garbage Collector)
- The C++-like ability to overload operators for a custom type, without the complexity
- Support for attribute-based programming. This brand of development allows you to annotate types and their members to further qualify their behavior. For example, if you mark a method with the [Obsolete] attribute, programmers will see your custom warning message print out if they attempt to make use of the decorated member.

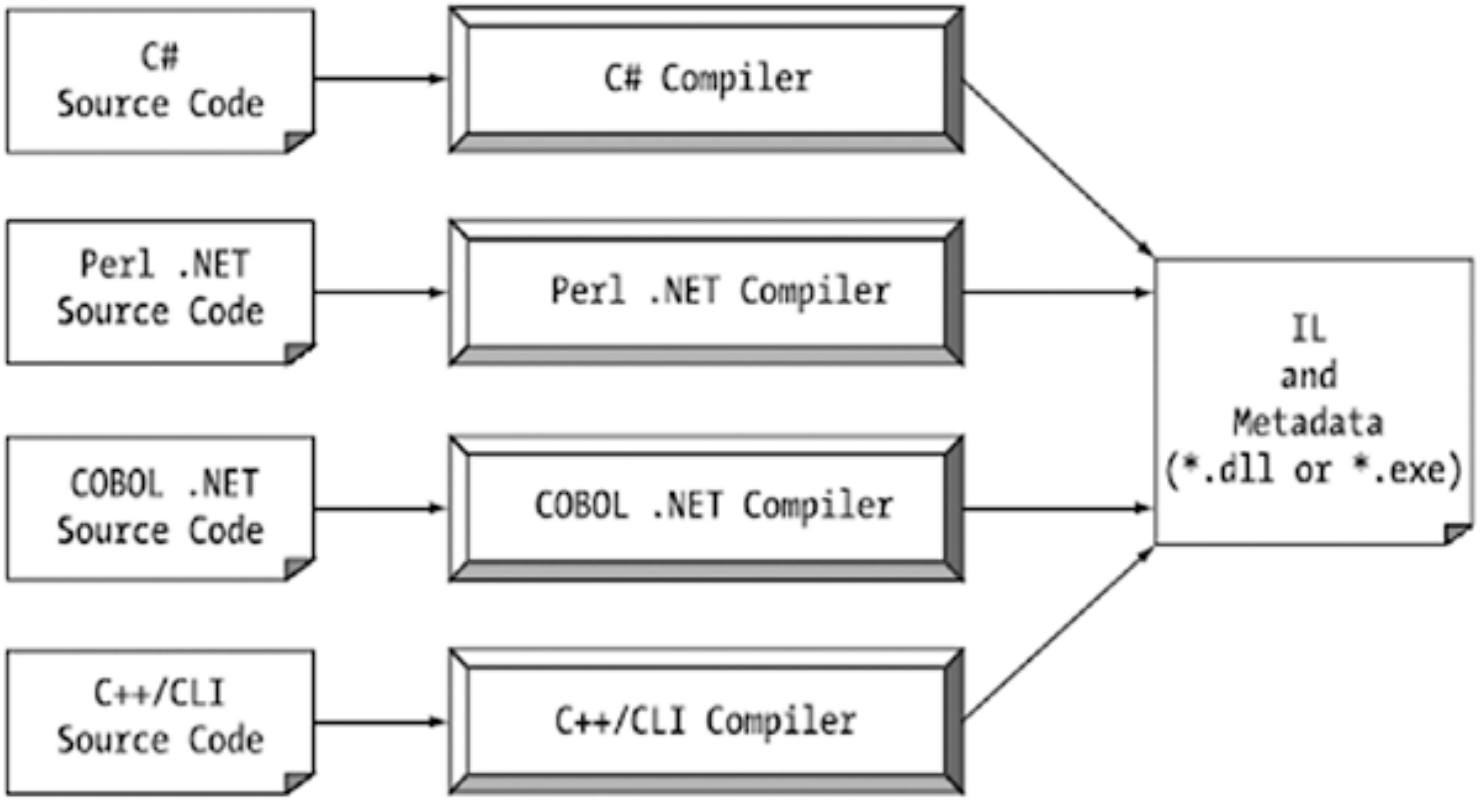
MANAGED VS. UNMANAGED CODE

- It is important to note that the C# language can be used only to build software that is hosted under the .NET runtime.
- The term used to describe the code targeting the .NET runtime is *managed code*. The binary unit that contains the managed code is termed an *assembly.*
- Conversely, code that cannot be directly hosted by the .NET runtime is termed *unmanaged code*.

AN OVERVIEW OF .NET ASSEMBLIES

- Regardless of which .NET language you choose to program with, understand that despite .NET binaries taking the same file extension as unmanaged Windows binaries (*.dll or *.exe), they have absolutely no internal similarities.
- Specifically, .NET binaries do not contain platform-specific instructions but rather platform-agnostic *Intermediate Language* (*IL*) and type metadata.

AN OVERVIEW OF .NET ASSEMBLIES



AN OVERVIEW OF .NET ASSEMBLIES

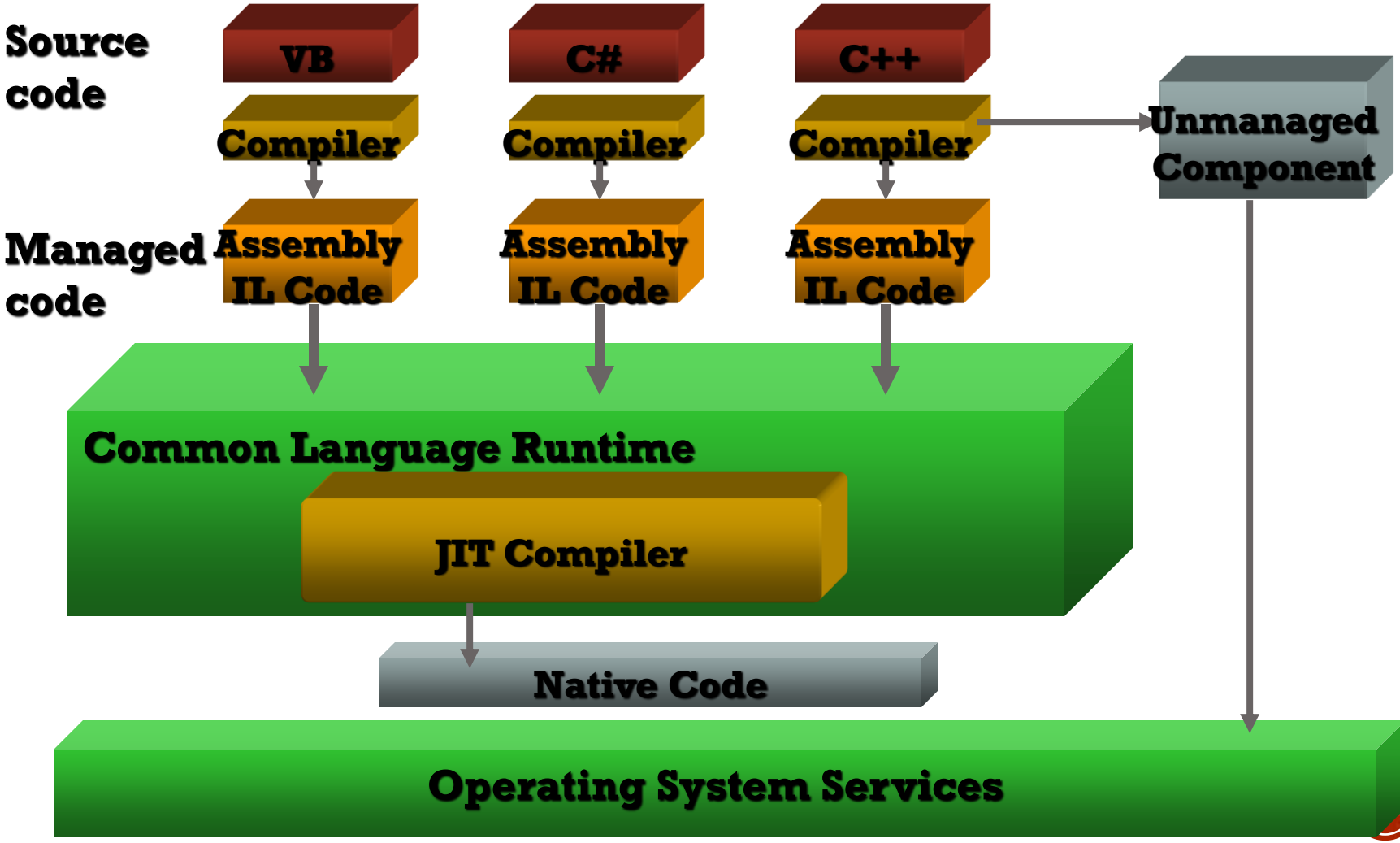
- When a *.dll or *.exe has been created using a .NET-aware compiler, the binary blob is termed an assembly.
- Similar to Java bytecode (i.e. not platform specific instructions)
- Assemblies contain:
  - MSIL (Microsoft Intermediate Language)
  - Metadata
  - Manifest

BENEFITS OF CIL/IL

- Why do we need CIL rather then directly converting to a specific instruction set?
  - Language integration: Each .NET-aware compiler produces nearly identical CIL instructions. Therefore, all languages are able to interact within a well-defined binary arena.
  - Platform-agnostic: a single codebase running on numerous operating systems.

.NET Framework and CLR

CLR Execution Model



Understanding the Common Type System

- A given assembly may contain any number of distinct types.
- In the world of .NET, *type* is simply a general term used to refer to a member from the set {class, interface, structure, enumeration, delegate}.

CTS CLASS TYPES

- Every .NET-aware language supports, at the least, the notion of a *class type*, which is the cornerstone of object-oriented programming (OOP). A class may be composed of any number of members (such as constructors, properties, methods, and events) and data points (fields).
- In C#, classes are declared using the class keyword, like so:

- ```
  class Calc
  {
    public int Add(int x, int y)
    {
      return x + y;
    }
  }
  ```


  ```

CTS CLASS TYPES

- Single inheritance
- Multiple interface implementation
- Static and instance members
- Nested types
- Member access
- Public, protected, private, internal, internal-protected, private-protected

CTS INTERFACE TYPES

- *Interfaces* are nothing more than a named collection of abstract member definitions, which may be supported (i.e., implemented) by a given class or structure.
- In C#, interface types are defined using the interface keyword.
- By convention, all .NET interfaces begin with a capital letter *I*, as in the following example:

- ```
  public interface Idraw
  {
   void Draw();
  }
  ```


  ```

CTS STRUCTURE TYPES

- A *structure* can be thought of as a lightweight class type having value-based semantics. Typically, structures are best suited for modeling geometric and mathematical data and are created in C# using the struct keyword, as follows:
- struct Point
```
{
  // Structures can contain fields.
  public int xPos, yPos;
  // Structures can contain parameterized constructors.
  public Point(int x, int y)
  { xPos = x; yPos = y;}
  // Structures may define methods.
  public void PrintPosition()
  {
  Console.WriteLine("({0}, {1})", xPos, yPos);
  }
}
```
- No inheritance

CTS Enumeration Types

- *Enumerations* are a handy programming construct that allow you to group name-value pairs.
- // A C# enumeration type.
  enum CharacterType
  {
  Wizard = 100,
  Fighter = 200,
  Thief = 300
  }
- By default, the storage used to hold each item is a 32-bit integer; however, it is possible to alter this storage slot if need be (e.g., when programming for a low-memory device such as a mobile device). Also, the CTS demands that enumerated types derive from a common base class, System.Enum.

CTS DELEGATE TYPES

- *Delegates* are the .NET equivalent of a type-safe, C-style function pointer.
- The key difference is that a .NET delegate is a *class* that derives from System.MulticastDelegate, rather than a simple pointer to a raw memory address.
- delegate int BinaryOp(int x, int y);
- Delegates are critical when you want to provide a way for one object to forward a call to another object and provide the foundation for the .NET event architecture.

LANGUAGE FEATURES
TYPE SYSTEM

- Value types
  - Directly contain data
  - Cannot be null
- Reference types
  - Contain references to objects
  - May be null

```
int i = 123;
string s = "Hello world";
```

LANGUAGE FEATURES
TYPE SYSTEM

- Value types
  - Primitives — `int i;`
  - Enums — `enum State { Off, On }`
  - Structs — `struct Point { int x, y; }`
- Reference types
  - Classes — `class Foo: Bar, IFoo {...}`
  - Interfaces — `interface IFoo: IBar {...}`
  - Arrays — `string[] a = new string[10];`
  - Delegates — `delegate void Empty();`

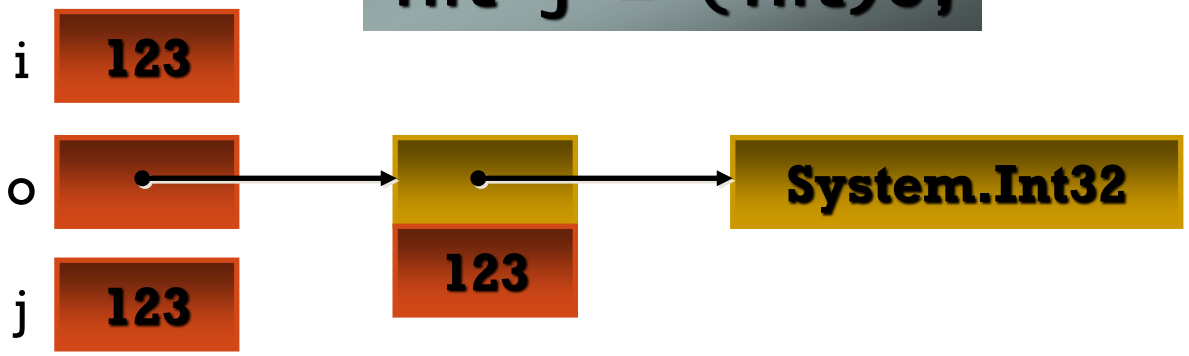
LANGUAGE FEATURES
PREDEFINED TYPES

- C# predefined types
  - Reference — object, string
  - Signed — sbyte, short, int, long
  - Unsigned — byte, ushort, uint, ulong
  - Character — char
  - Floating-point — float, double, decimal
  - Logical — bool
- Predefined types are simply aliases for system-provided types
  - For example, int = System.Int32

LANGUAGE FEATURES
UNIFIED TYPE SYSTEM

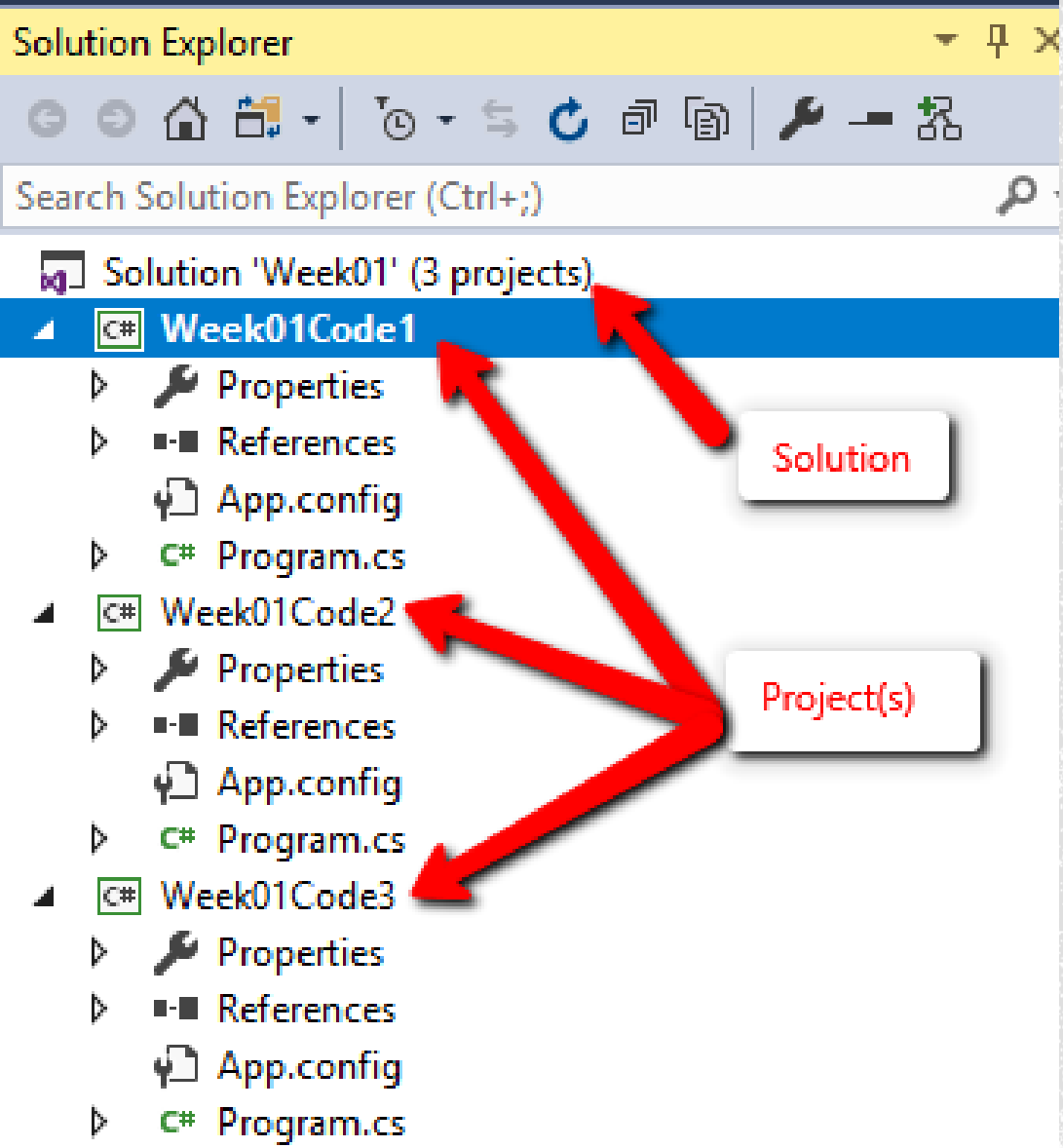
- Boxing
  - Allocates box, copies value into it
- Unboxing
  - Checks type of box, copies value out

```
int i = 123;
object o = i;
int j = (int)o;
```



NAMESPACE

- A namespace is a grouping of semantically related types contained in an assembly or possibly spread across multiple related assemblies.
- For example, the System.IO namespace contains file I/O–related types, the System.Data namespace defines basic database types, and so on.
- It is important to point out that a single assembly (such as mscorlib.dll) can contain any number of namespaces, each of which can contain any number of types.



SOLUTION V/S PROJECT

RECAP OF OOP

- Pillars of Object-Oriented Programming:
  - Inheritance
  - Encapsulation
  - Polymorphism
  - Abstraction

ACCESS SPECIFIERS

- Public
- Private
- Protected
- Internal
- Protected Internal
- Private Protected

C# Access Modifiers				
	Same Assembly		Different Assembly	
	Child Class	Via Object	Child Class	Via Object
Public	Y	Y	Y	Y
Private	N	N	N	N
Protected	Y	N	Y	N
Internal	Y	Y	N	N
Protected Internal	Y	Y	Y	N
Private Protected	Y	N	N	N