ASP.NET

Unit-1

Unit-1 **ASP.NET Introduction** **(Marks 18)**

- **The .Net Framework 4.5**
- **Feature of .Net ,The .Net Languages**
- **CLR**
- **.Net Class Library, Object of ASP**
- **Data Types, Variable & Expression, Scope & Accessibility, Looping Statement**
- **Delegates**

❖ ASP.NET Introduction

- ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites, web applications and web services.
- ASP.NET, the next version of ASP, is a programming framework used to create enterprise-class Web Applications.
- ASP.NET (originally called ASP+) is the next generation of Microsoft's Active Server Page (ASP),
- It was first released in January 2002 with version 1.0 of the .NET Framework
- ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language.
- ASP.NET allows you to use a full featured programming language such as C# (pronounced C-Sharp) or VB.NET to build web applications easily.
- Both ASP and ASP.NET allow a Web site builder to dynamically build Web pages by inserting queries to a relational database in the Web page.
- ASP.NET files can be recognized by their .aspx extension.

Advantages Using ASP.NET

- ASP.NET drastically reduces the amount of code required to build large applications
- ASP.NET makes development simpler and easier to maintain with an event-driven, server-side programming model
- ASP.NET pages are easy to write and maintain because the source code and HTML are together
- The source code is executed on the server. The pages have lots of power and flexibility by this approach

- The source code is compiled the first time the page is requested. Execution is fast as the Web Server compiles the page the first time it is requested. The server saves the compiled version of the page for use next time the page is requested.
- The HTML produced by the ASP.NET page is sent back to the browser. The application source code you write is not sent and is not easily stolen.
- ASP.NET makes for easy deployment. There is no need to register components because the configuration information is built-in.
- The Web server continuously monitors the pages, components and applications running on it. If it noticies memory leaks, infinite loops, other illegal software or activities, it seamlessly kills those activities and restarts itself.
- ASP.NET validates information (validation controls) entered by the user without writing a single line of code.
- ASP.NET easily works with ADO .NET using data-binding and page formatting features.
- ASP.NET applications run faster and counters large volumes of users without performance problems

Differences between ASP.NET and Client-Side Technologies

- Client-side refers to the browser and the machine running the browser.
- Server-side on the other hand refers to a Web server.

Client-Side Scripting

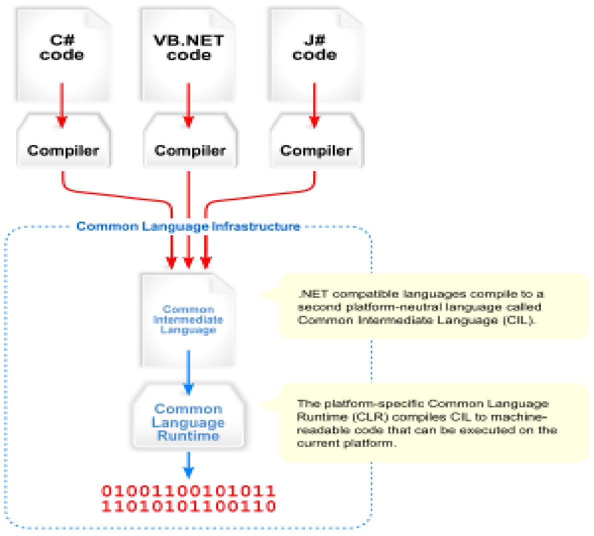
Javascript and VBScript and generally used for Client-side scripting. Client-side scripting executes in the browser after the page is loaded. Using client-side scripting you can add some cool features to your page. Both, HTML and the script are together in the same file and the script is download as part of the page which anyone can view. A client-side script runs only on a browser that supports scripting and specifically the scripting language that is used. Since the script is in the same file as the HTML and as it executes on the machine you use, the page may take longer time to download.

Server-Side Scripting

ASP.NET is purely server-side technology. ASP.NET code executes on the server before it is sent to the browser. The code that is sent back to the browser is pure HTML and not ASP.NET code. Like client-side scripting, ASP.NET code is similar in a way that it allows you to write your code alongside HTML. Unlike client-side scripting, ASP.NET code is executed on the server and not in the browser. The script that you write alongside your HTML is not sent back to the browser and that prevents others from stealing the code you developed.

❖ The .Net Framework 3.0

- The Microsoft .NET Framework is a software framework that can be installed on computers that running Microsoft Windows operating systems.
- It includes a large library of coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework.


C# code
VB.NET code
J# code
Compiler
Compiler
Compiler
Common Language Infrastructure
Common Intermediate Language
.NET compatible languages compile to a second platform-neutral language called Common Intermediate Language (CIL).
Common Language Runtime
The platform-specific Common Language Runtime (CLR) compiles CIL to machine-readable code that can be executed on the current platform.
01001100101011
11010101100110

- The .NET framework supports multiple programming languages in a manner that allows language interoperability, whereby each language can utilize code written in other languages
- The framework's Base Class Library provides a large range of features including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications.
- The class library is used by programmers, who combine it with their own code to produce applications.
- Programs written for the .NET Framework execute in a software environment that **<u>manages the program's runtime requirements.</u>**
- This runtime environment is known as the Common Language Runtime (CLR).

.Net Framework Versions:

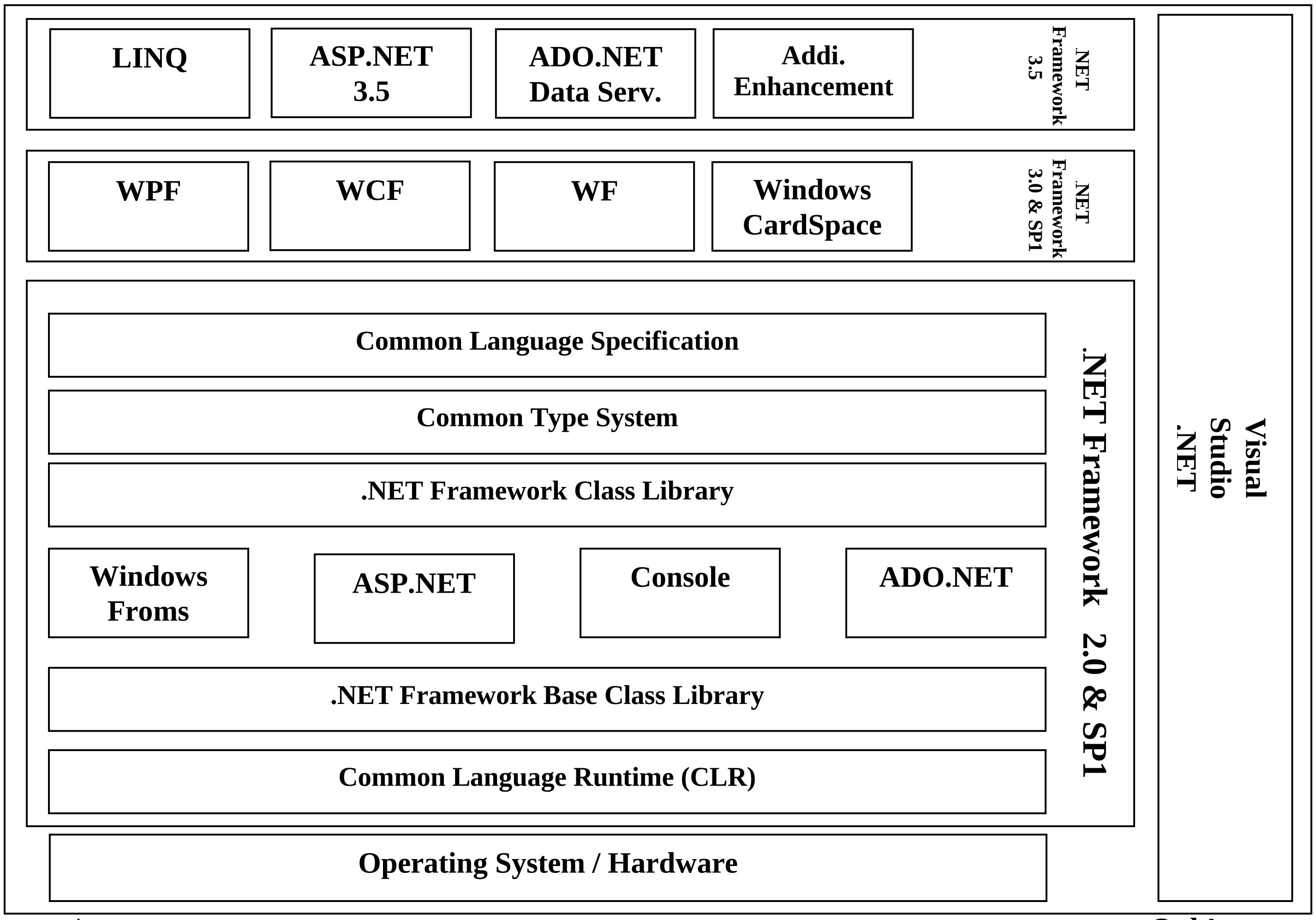
- Microsoft started development on the .NET Framework in the late 1990s originally under the name of Next Generation Windows Services (NGWS).

Version	Version Number	Release Date	Visual Studio	Default in Windows
1.0	1.0.3705.0	2002-02-13	Visual Studio .NET	
1.1	1.1.4322.573	2003-04-24	Visual Studio .NET 2003	Windows Server 2003
2.0	2.0.50727.42	2005-11-07	Visual Studio 2005	
3.0	3.0.4506.30	2006-11-06		Windows Vista, Windows Server 2008
3.5	3.5.21022.8	2007-11-19	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	2010-04-12	Visual Studio 2010	

.Net Framework 3.0

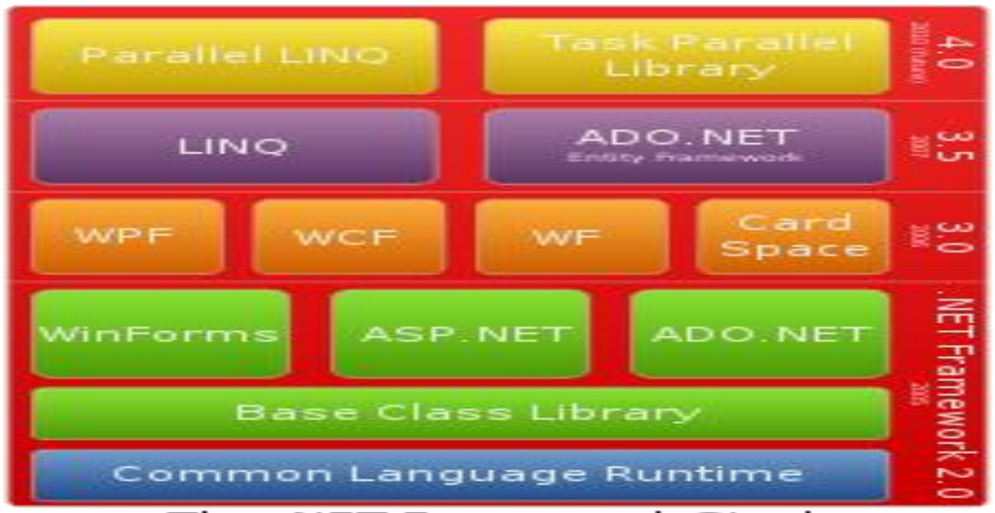
- It includes a new set of managed code APIs that are an integral part of Windows Vista and Windows Server 2008 operating systems.
- It is also available for Windows XP SP2 and Windows Server 2003
- .NET Framework 3.0 uses the Common Language Runtime of .NET Framework 2.0

.Net Framework Architecture



.NET Framework 3.0 Consists of Four Major New Components:

- **Windows Presentation Foundation (WPF)** A new user interface subsystem and API based on XML and vector graphics, which uses 3D computer graphics hardware and Direct3D technologies
- **Windows Communication Foundation (WCF)** A service-oriented messaging system which allows programs to interoperate locally or remotely similar to web services.
- **Windows Workflow Foundation (WF)** Allows for building of task automation and integrated transactions using workflows.
- **Windows CardSpace** A software component which securely stores a person's digital identities and provides a unified interface for choosing the identity for a particular transaction, such as logging in to a website.



The .NET Framework Stack

Additional

❖ Components of .Net Framework 3.0

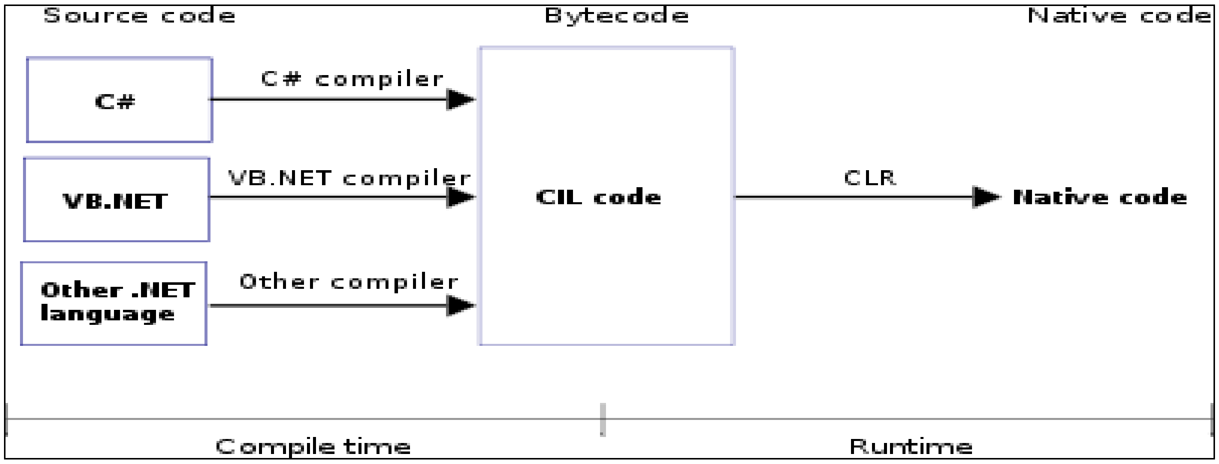
- Common Language Runtime. (CLR)
- .NET Framework Class Library. (FCL)
- Common Language Specification. (CLS)
- Common Type System. (CTS)
- Metadata and Assemblies.
- Manage Code and Umanage Code.
- Windows Forms.
- ASP.NET and ASP.NET AJAX.
- ADO.NET
- Windows Presentation Foundation. (WPF)
- Windows Communication Foundation. (WDF)
- Windows WorkFlow Foundation. (WF).
- Windows CardSpace.

❖ Common Language Runtime. (CLR)

- The .NET Framework provides a run-time environment called the common language runtime.
- CLR runs the code and provides services that make the development process easier.
- The Common Language Runtime manages the execution of the code.
- It provides the functionalities, such as memory management, exception handling, debugging, security, thread execution, code execution, code safety, verification, compilation.
- The code that works on the CLR is called the managed code.
- The CLR automatically release the objects when they are no logner in use. This automatically memory management is known as Garbage Collection.
- It is Microsoft's implementation of the Common Language Infrastructure (CLI) standard.

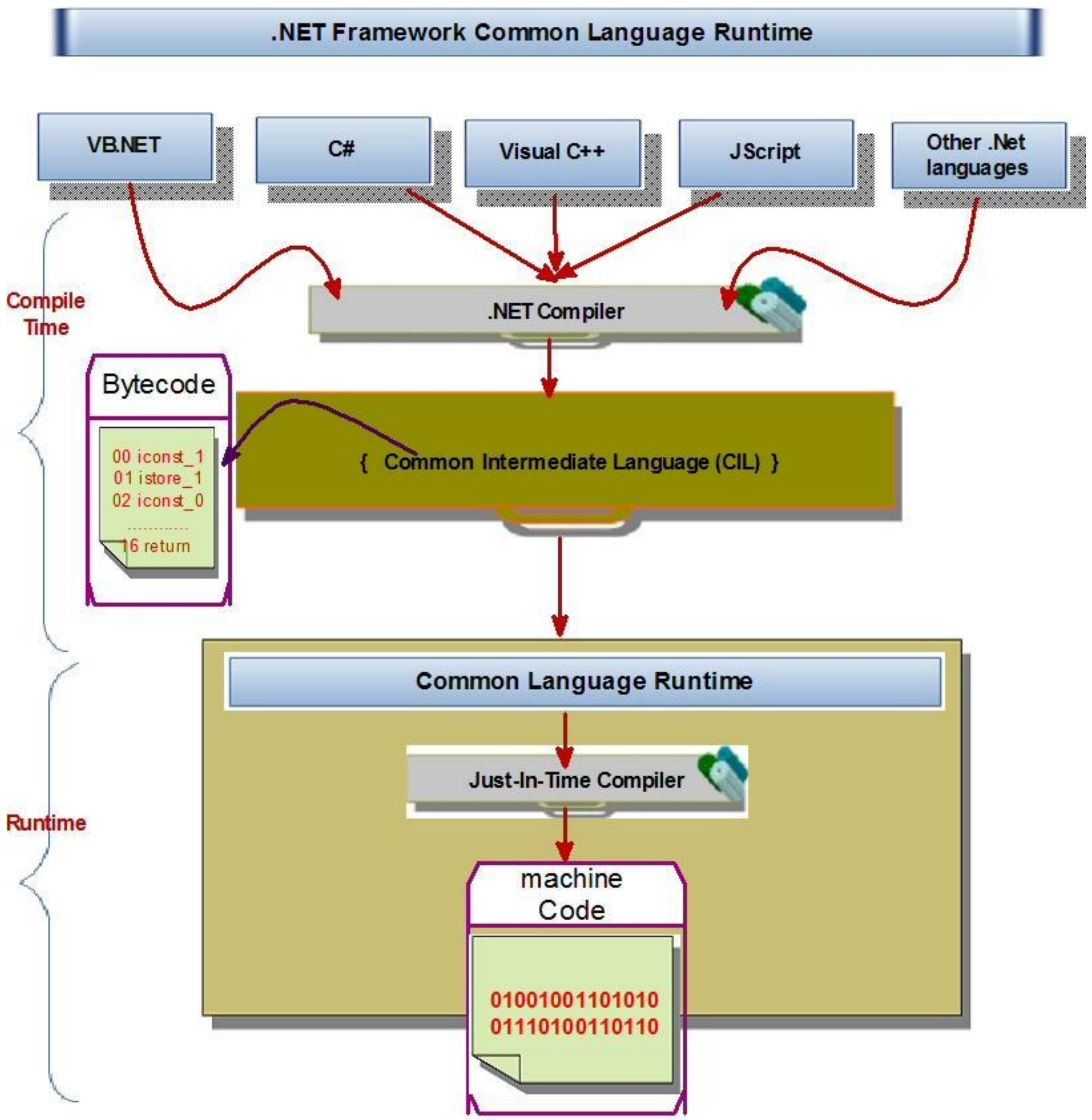
- Developers using the CLR write code in a language such as C# or VB.NET.

  - **At compile time**, a .NET compiler converts such code into CIL code.
  - **At runtime**, the CLR's just-in-time compiler converts the CIL code into code native to the operating system.



- A Piece of Managed code is executed as follows:

  1. Choosing language compiler.
  2. Compiling the code to IL.
  3. Compiling IL to native code.
  4. Executing the code.



- When you compile the code into managed code, the compiler converts the source code into IL(Intermediate Languages OR CIL (common)),
- Before the execution of the code, IL must be converted into CPU-Specific code by the JUST-IN-TIME (JIT) compiler.
- When you compile your source code into IL, the required metadata is generated.
- Before running the IL, it must be converted to native code. The native code is CPU-specific code which is nothing but the output of the JIT compiler, that is, the machine code that actually runs in the runtime.

❖ Manage Code.

- Manage code is code that is executed directly by the CLR.
- The application created using managed code automatically have CLR services, such as type checking, security, automatically garbage collection.
- This IL along with the metadata that describes the attributes, classes and methods of the code.
- The CLR compiles the application to Intermediate Language (IL) and not the machine code.

❖ Unmanage Code.

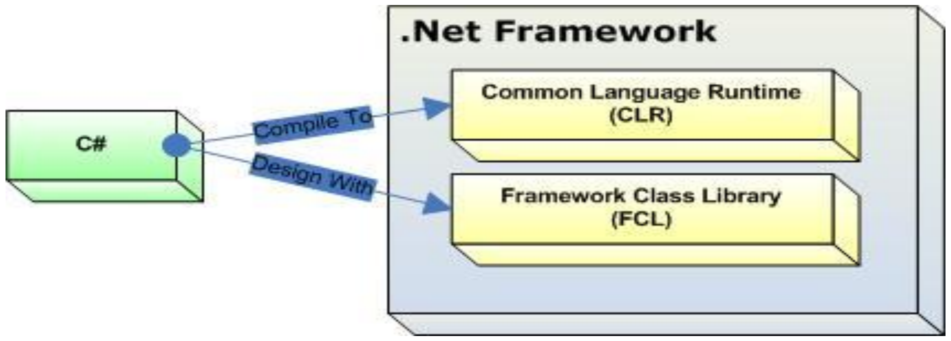
- In contrast to the managed code, unmanaged code directly compiles to the machine code and runs on the machine where it has been compiled.
- It does not have services , such as security or memory management, which are provided by the CLR.
- If your code is not security-prone, it can be directly interpreted (or Manipulated) by any user, which can prove harmful.

❖ Memory Management.

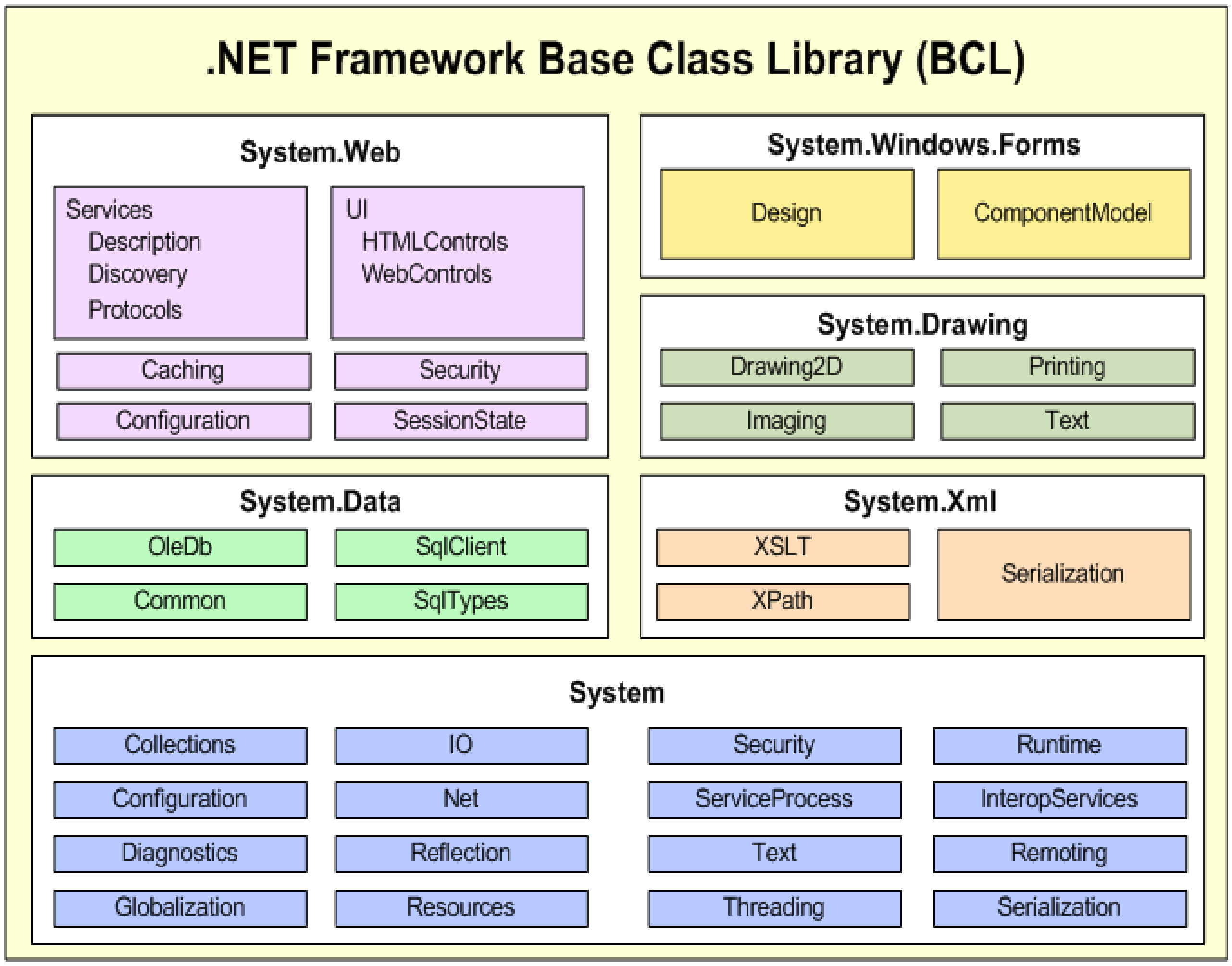
- One of the most important services that the CLR provides during managed execution is the automatic memory management.
- The CLR uses garbage collector to manage the allocation and release of memory for an application.
- Automatic memory management removes common problems, such as forgetting to free an object which results in memory leak, or attempting to access memory for an object that is already free.

❖ .NET Framework Class Library. (FCL)

- The .NET Framework class library is a library of classes, interfaces, and value types that are included in the Microsoft .NET Framework SDK.



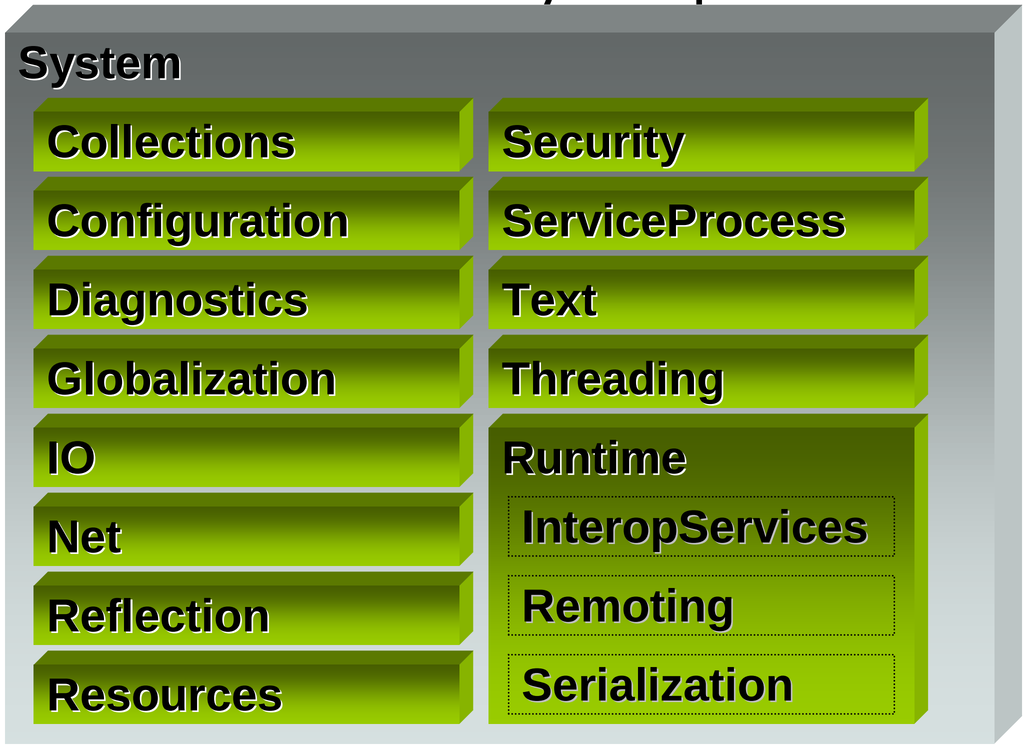
- This library provides access to system functionality and is designed to be the foundation on which .NET Framework applications, components, and controls are built.
- The .Net Framework class library (FCL) organized in a hierarchical tree structure and it is divided into **Namespaces.**



.NET Framework Namespaces

- Namespaces is a logical grouping of types for the purpose of identification. Framework class library (FCL) provides the consistent base types that are used across all .NET enabled languages.
- The Classes are accessed by namespaces, which reside within Assemblies. The System Namespace is the root for types in the .NET Framework.
- The .Net Framework class library (FCL) classes are managed classes that provide access to System Services . The .Net Framework class library (FCL) classes are object oriented and easy to use in program developments. Moreover, third-party components can integrate with the classes in the .NET Framework.

Base Class Library Namespaces



- Data types, conversions, formatting
- Collections: ArrayList, Hashtable, etc.
- Globalization: Cultures, sorting, etc.
- I/O: Binary and text streams, files, etc.
- Networking: HTTP, TCP/IP sockets, etc.
- Reflection: Metadata and IL
- Security: Permissions, cryptography
- Text: Encodings, regular expressions

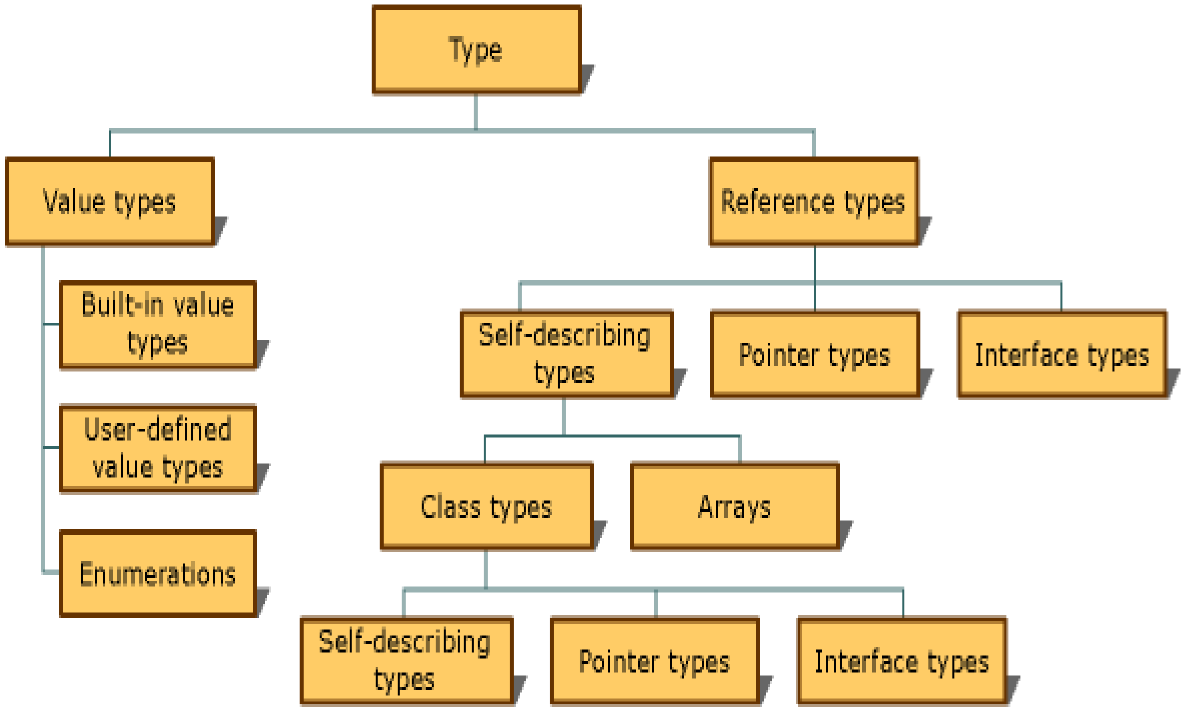
❖ Common Type System. (CTS)

- The common type system defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration.
- The common type system performs the following functions:

  - Establishes a framework that helps enable cross-language integration, type safety, and high performance code execution.
  - Provides an object-oriented model that supports the complete implementation of many programming languages.

- Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.

- The common type system supports two general categories of types
    - Value types
    - Reference types



Common Type System(CTS).

Value types

- Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure.
- Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

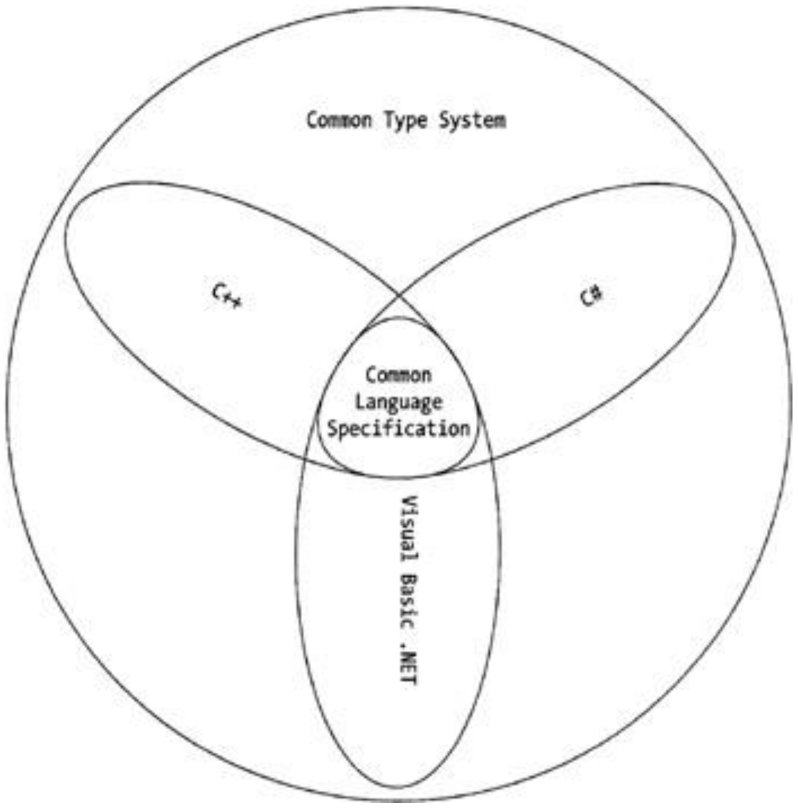
Reference types

- Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types.

- he type of a reference type can be determined from values of self-describing types.
- Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

❖ Common Language Specification. (CLS)

- The Common Language Specification (CLS), which is a set of basic language features needed by many .Net applications to fully interact with other objects regardless of the language in which they were implemented.
- This is a subset of the CTS which all .NET languages are expected to support.



CLS

- Most of the members defined by types in the .NET Framework Class Library (FCL) are Common Language Specification (CLS) compliant Types.
-

❖ Intermediate Language

- MSIL stands for Microsoft Intermediate Language. We can call it as Intermediate Language (IL) or Common Intermediate Language (CIL).
- During the compile time , the compiler convert the source code into Microsoft Intermediate Language (MSIL) .Microsoft Intermediate Language (MSIL) is a CPU-independent set of instructions that can be efficiently converted to the native code. During the runtime the Common Language Runtime (CLR)'s Just In Time (JIT) compiler converts the Microsoft Intermediate Language (MSIL) code into native code to the Operating System.
- When a compiler produces Microsoft Intermediate Language (MSIL), it also produces Metadata. The Microsoft Intermediate Language (MSIL) and Metadata are contained in a portable executable (PE) file .
- Microsoft Intermediate Language (MSIL) includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations.

❖ Portable Executable (PE) File

- The Portable Executable (PE) format is a file format for executables, object code, and DLLs, used in 32-bit and 64-bit versions of Windows operating systems.
- The PE file format was defined to provide the best way for the Windows Operating System to execute code and also to store the essential data which is needed to run a program.

❖ Just In Time Compiler

- The .Net languages, which is conforms to the Common Language Specification (CLS), uses its corresponding runtime to run the application on different Operating Systems . During the code execution time, the Managed Code compiled only when it is needed, that is it converts the appropriate instructions to the native code for execution just before when each function is called. This process is called Just In Time (JIT) compilation, also known as Dynamic Translation.

❖ .Net Assembly

- Microsoft .Net Assembly is a logical unit of code, it contains code that the Common Language Runtime (CLR) executes.
- Assembly is really a collection of types and resource information that are built to work together and form a logical unit of functionality.
- During the compile time Metadata is created, with Microsoft Intermediate Language (MSIL), and stored in a file called a Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file.
- Manifest contains information about itself. This information is called Assembly Manifest, it contains information about the members, types, references and all the other data that the runtime needs for execution.
- Every Assembly you create contains one or more program files and a Manifest. There are two types program files : Process Assemblies (EXE) and Library Assemblies (DLL).

❖ Garbage Collection

- The .Net Framework provides a new mechanism for releasing unreferenced objects from the memory (that is we no longer needed that objects in the program), this process is called Garbage Collection (GC).
- When a program creates an Object, the Object takes up the memory. Later when the program has no more references to that Object, the Object's memory becomes unreachable, but it is not immediately freed. The Garbage Collection checks to see if there are any Objects in the heap that are no longer being used by the application. If such Objects exist, then the memory used by these Objects can be reclaimed.

Data Types:

Type	Represents
Bool	Boolean value
Byte	8-bit unsigned integer
Char	16-bit Unicode character
Decimal	28-bit precise decimal values with 28-29 significant digit
Double	64-bit double-precision floating point type
Float	32-bit single-precision floating point type
Int	32-bit signed integer type
Long	64-bit signed integer type
Sbyte	8-bit signed integer type
Short	16-bit signed integer type
Uint	32-bit unsigned integer type
Ulong	64-bit unsigned integer type
Ushort	16-bit unsigned integer type

Variable & Expression.

- As with all programming languages, you keep track of data in C# by using variables.
- Variables can store numbers, text, dates, and times, and they can even point to full-fledged objects.
- When you declare a variable, you give it a name and specify the type of data it will store.
- To declare a local variable, you start the line with the data type, followed by the name you want to use.
- A final semicolon ends the statement.

Scope & Accessibility

- If you define a variable inside some sort of block structure (such as a loop or a conditional block)
- The variable is automatically released when your code exits the block.
- That means you will no longer be able to access it.
- The following code demonstrates this behavior:

  Example:

```
int tempVariableA;
for (int i = 0; i < 10; i++)
        {
                int tempVariableB;
                tempVariableA = 1;
                tempVariableB = 1;
        }
```

- You cannot access tempVariableB here
- However, you can still access tempVariableA

Looping Statement.

- Loops allow you to repeat a segment of code multiple times.
- C# has three basic types of loops.
- You choose the type of loop based on the type of task you need to perform.
- **Your choices are as follows:**
- You can loop a set number of times with a **for loop**.
- You can loop through all the items in a collection of data by using a **foreach loop**.
- You can loop while a certain condition holds true with **a while** or **do…while loop**.

The for Loop

- The for loop is a basic ingredient in many programs.
- It allows you to repeat a block of code a set number of times, using a built-in counter
- To create a for loop, you need to specify a starting value, an ending value, and the amount to increment with each pass.
- **Example:**

```
for (int i = 0; i < 10; i++)
    {
        // This code executes ten times.
        System.Diagnostics.Debug.Write(i);
      }
```

The foreach Loop

- C# also provides a foreach loop that allows you to loop through the items in a set of data
- With a foreach loop, you don’t need to create an explicit counter variable
- Instead, you create a variable that represents the type of data for which you’re looking.

- Your code will then loop until you've had a chance to process each piece of data in the set.
- The foreach loop is particularly useful for traversing the data in collections and arrays.
- Example

```
int[] intArray = {1,2,3};
foreach (int num in intArray)
{
    num += 1;
}
```

The while loop

At the beginning of each pass, the code evaluates whether the counter (i) is less than some upper limit (in this case, 10). If it is, the loop performs iteration.

Example:1

```
int i = 0;
while (i < 10)
{
    i += 1;
    // This code executes ten times.
}
```

Example:2

```
int i = 0;
do
    {
        i += 1;
        // This code executes ten times.
    }
while (i < 10);
```

Delegates

- C# delegates are similar to pointers to functions, in C or C++.
- A delegate is a reference type variable that holds the reference to a method.
- The reference can be changed at runtime.
- Delegates are especially used for implementing events and the call-back methods.
- All delegates are implicitly derived from the System.Delegate class.

Declaring Delegates

- Delegate declaration determines the methods that can be referenced by the delegate.
- A delegate can refer to a method, which have the same signature as that of the delegate.

For example, consider a delegate:

```
public delegate int MyDelegate (string s);
```