Week 01:

- **INTRODUCTION**
  - Overview of c#
  - C# family tree
  - C# and **.**NET Framework
  - Version History of C#

Introduction

Overview of C#

C# is pronounced as "See Sharp". It is a simple, modern, object-oriented, and type-safe programming language. C# has its roots in the C family of languages. It enables developers to build a variety of secure and robust applications that run on the .NET Framework. C# can be used to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more.

Several C# features aid in the construction of robust and durable applications:

- ***Garbage collection*** automatically reclaims memory occupied by unreachable unused objects
- ***Exception handling*** provides a structured and extensible approach to error detection and recovery
- ***Type-safe*** design of the language makes it impossible to read from uninitialized variables, to index arrays beyond their bounds, or to perform unchecked type casts
- ***Unified type system***: All C# types, including primitive types such as ***int*** and ***double***, inherit from a single root *object* type. Thus, all types share a set of common operations, and values of any type can be stored, transported, and operated upon in a consistent manner.
- C# supports both user-defined ***reference types*** and ***value types***, allowing dynamic allocation of objects as well as in-line storage of lightweight structures.
- ***Versioning***ensure that C# programs and libraries can evolve over time in a compatible manner

C# is an object-oriented language, but C# further includes support for ***component-oriented***programming.As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism. All variables and methods, including the Main method (the application's entry point) are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. Methods that override virtual methods in a parent class require the override keyword as a way to

avoid accidental redefinition. In C#, a struct is like a lightweight class; it is a stack-allocated type that can implement interfaces but does not support inheritance.

In addition to these basic object-oriented principles, C# makes it easy to develop software components through several innovative language constructs, including the following:

- Encapsulated method signatures called *delegates*, which enable type-safe event notifications.
- Properties, which serve as accessors for private member variables.
- Attributes, which provide declarative metadata about types at run time.
- Inline XML documentation comments.
- Language-Integrated Query (LINQ) which provides built-in query capabilities across a variety of data sources.

C# syntax simplifies many of the complexities of C++ and provides powerful features such as

- nullable value types, enumerations, delegates, lambda expressions and direct memory access, which are not found in Java.

C# supports

- generic methods and types, which provide increased type safety and performance
- iterators, which enable implementers of collection classes to define custom iteration behaviors that are simple to use by client code

Visual C# provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to make it easier to develop applications based on the C# language and the .NET Framework.

C# Family Tree

C# inherits a rich programming legacy. It is directly descended from two of the world's most successful computer languages: C and C++. It is closely related to Java.

Invention of C language is considered as the beginning of the modern age of programming. C was invented by Dennis Ritchie in the 1970s on a DEC PDP-11 that used the UNIX operating system. By the late 1970s, the size of many projects was near or at the limits of what structured programming methodologies and the C language could handle.

Using OOP, a programmer could handle much larger programs. The desire for an object-oriented version of C led to the creation of C++beginning in 1979.C++ was invented by BjarneStroustrup at Bell Laboratories in Murray Hill, New Jersey. C is the foundation upon which C++ is built. In essence, C++ is the object-oriented version of C. By building upon the foundation of C, Stroustrup provided a smooth migration path to OOP. Instead of having to learn an entirely new language, a C programmer needed to learn only a few new features before reaping the benefits of the object-oriented methodology. This made it easy for legions of programmers to make the shift from structured programming to objectoriented programming.It was an enhancement to an already highly successful language.

The main driving force behind Java's design was James Gosling. Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan also played a role. Java is a structured, object-oriented language with a syntax and philosophy derived from C++. The innovative aspects of Java were driven not so much by advances in the art of programming, but rather by changes in the computing environment

Java achieved portability by translating a program's source code into an intermediate language called ***bytecode***. This bytecode was then executed by the Java Virtual Machine (JVM). Therefore, a Java program could run in any environment for which a JVM was available.

It is key to understand that Java's use of bytecode differed radically from both C and C++, which were nearly always compiled to executable machine code. Machine code is tied to a specific CPU and operating system. Thus, if you wanted to run a C/C++ program on a different system, it needed to be recompiled to machine code specifically for that environment, to create a C/C++ program that would run in a variety of environments.

Creation of C#

One of the lacks of Java is *cross-language interoperability* also called *mixed-language programming*. Cross-language interoperability is crucial for the creation of large, distributed software systems. Another feature lacking in Java is full integration with the Windows platform. Java and Windows are not closely coupled. Since Windows is the mostly widely used operating system in the world, lack of direct support for Windows is a drawback to Java.To answer these and other needs, Microsoft developed C#.

C# was created at Microsoft late in the 1990s and was part of Microsoft's overall .NET strategy.By building C# upon a solid,well-understood foundation, C# offers an easy migration path from these languages. Some of its most important features relate to its built-in support for softwarecomponents. Infact,C# has been characterized as being a component-oriented language because it contains integral support for the writing of software components.

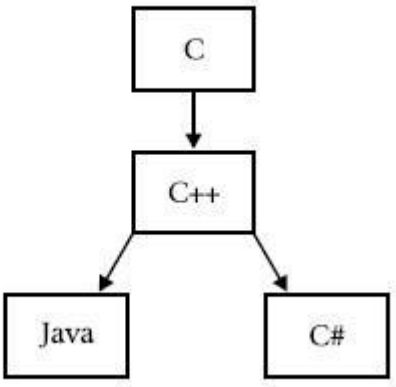


Figure 1: The family tree for C#

C# and .NET Framework

C# programs run on the .NET Framework. Dot net (.NET) framework is an integral component of Windows that includes

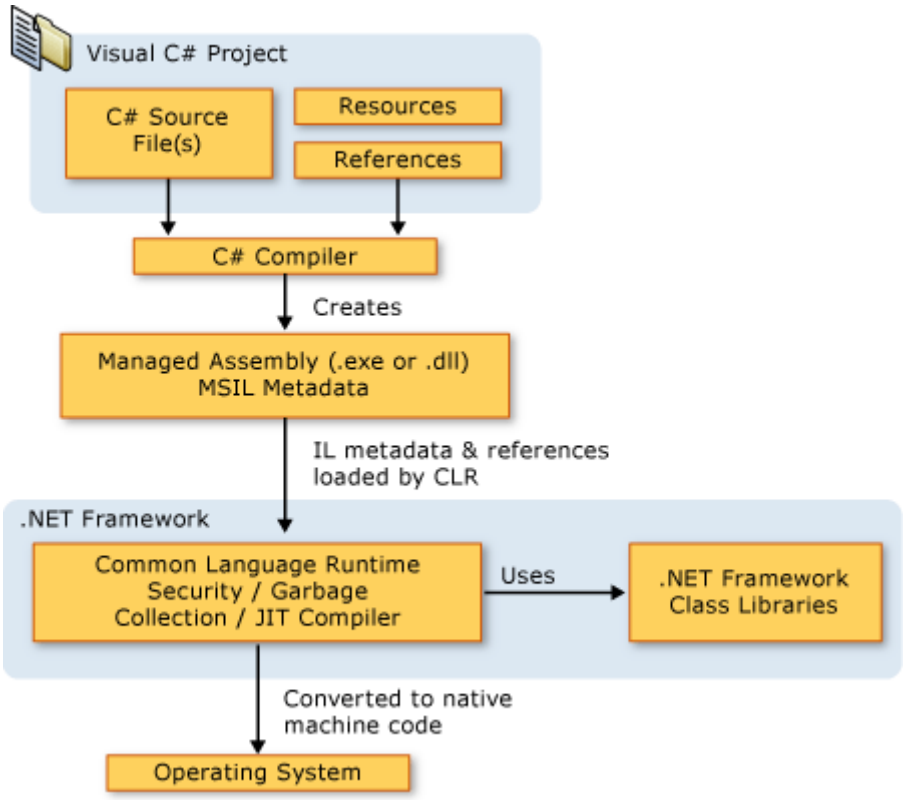
- a virtual execution system called the common language runtime (CLR) and
- a unified set of class libraries

The CLR is the commercial implementation by Microsoft of the **common language infrastructure** (CLI). CLI is an international standard that is the basis for creating execution and development environments in which languages and libraries work together.

Source code written in C# is compiled into an ***intermediatelanguage*** (**IL**) that conforms to the CLI specification. The **IL** code and resources, such as bitmaps and strings, are stored on disk in an executable file called an assembly, typically with an extension of .exe or .dll. An assembly contains a manifest that provides information about the assembly's types, version, culture, and security requirements.

When the C# program is executed, the assembly is loaded into the CLR, which might take various actions based on the information in the manifest. Then, if the security requirements are met, the CLR performs just in time (JIT) compilation to convert the IL code to native machine instructions. The CLR also provides other services related to automatic garbage collection, exception handling, and resource management. Code that is executed by the CLR is sometimes referred to as "managed code," in contrast to "unmanaged code" which is compiled into native machine language that targets a specific system.

The following diagram illustrates the compile-time and run-time relationships of C# source code files, the .NET Framework class libraries, assemblies, and the CLR.



Version History of C#

Version	.NET Framework	Visual Studio	New Features
C# 1.0	.NET Framework 1.0/1.1	Visual Studio .NET 2002	♦ Basic features
C# 2.0	.NET Framework 2.0	Visual Studio 2005	♦ Generics ♦ Partial types ♦ Anonymous methods ♦ Iterators ♦ Nullable types ♦ Private setters (properties) ♦ Method group conversions (delegates) ♦ Covariance and Contra-variance ♦ Static classes

Version	.NET Framework	Visual Studio	New Features
C# 3.0	.NET Framework 3.0\3.5	Visual Studio 2008	♦ Implicitly typed local variables ♦ Object and collection initializers ♦ Auto-Implemented properties ♦ Anonymous types ♦ Extension methods ♦ Query expressions ♦ Lambda expressions ♦ Expression trees ♦ Partial Methods
C# 4.0	.NET Framework 4.0	Visual Studio 2010	♦ Dynamic binding (late binding) ♦ Named and optional arguments ♦ Generic co- and contravariance ♦ Embedded interop types
C# 5.0	.NET Framework 4.5	Visual Studio 2012/2013	♦ Asynchronous methods ♦ Caller info attributes
C# 6.0	.NET Framework 4.6	Visual Studio 2013/2015	♦ Expression-bodied members ♦ Auto-property initializer ♦ *nameof* operator ♦ Primary constructor ♦ Await in catch/finally blocks ♦ Exception filters ♦ String interpolation ♦ Compiler-as-a-service ♦ Import of static type members into namespace ♦ Default values for getter-only properties

Version	.NET Framework	Visual Studio	New Features
			♦ Null propagator ♦ Dictionary initializer
C# 7.0	.NET Core .NET Framework 4.6.2	Visual Studio 2017	♦ out variables ♦ Tuples ♦ Discards ♦ Pattern Matching ♦ Local functions ♦ Generalized async return types ♦ throw Expressions ♦ Deconstruction ♦ Digit separators ♦ Binary literals ♦ Ref returns and locals ♦ Expression bodied constructors and finalizers ♦ Expression bodied getters and setters ♦ Use of Throw as expression
C# 7.1 August 2017	.Net Framework 4.7	Visual Studio 2017 version 15.3	♦ Async main ♦ Default literal expressions ♦ Inferred tuple element names
C# 7.2 November 2017	.Net Framework 4.7.1	Visual Studio 2017 version 15.5	♦ Reference semantics with value types ♦ Non-trailing named arguments ♦ Leading underscores in numeric literals ♦ private protected access modifier
C# 7.3 May 2018	.NET Core 2.1 .NET Core 2.2 .NET Framework 4.8	Visual Studio 2017 version 15.7	♦ Accessing fixed fields without pinning ♦ Reassigning ref local variables

Version	.NET Framework	Visual Studio	New Features
			♦ Using initializers on stackalloc arrays ♦ Using fixed statements with any type that supports a pattern ♦ Using additional generic constraints
C# 8.0 September 2019	.NET Core 3.0	Visual Studio 2019 version 16.3	♦ `readonlystruct` members ♦ Default interface members ♦ `switch` expressions ♦ Property, Tuple, and positional patterns ♦ `using` declarations ♦ `static` local functions ♦ Disposable `ref struct` ♦ Nullable reference types ♦ Indices and Ranges ♦ Null-coalescing assignment ♦ Async Streams

References:

- ✔ https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework
- ✔ https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
- ✔ https://www.wideskills.com/csharp/overview-csharp
- ✔ https://sites.google.com/site/progammingguidec/fundamentals/c-family-tree
- ✔ https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
- ✔ https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8

- ✔ https://sites.google.com/site/progammingguidec/fundamentals/c-family-tree/Untitled.jpg?attredirects=0
- ✔ https://docs.microsoft.com/en-us/dotnet/standard/managed-code