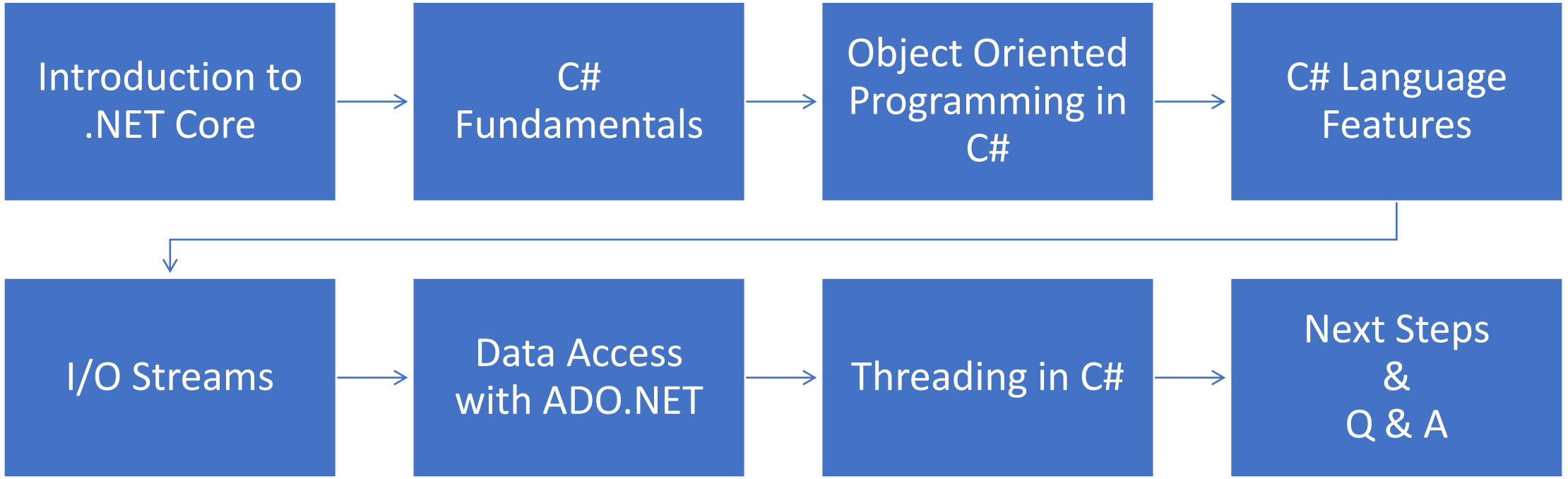
.NET Core and C# Fundamentals

By Anil Joseph

Timmins Training Consulting

Agenda


Introduction to .NET Core
C# Fundamentals
Object Oriented Programming in C#
C# Language Features
I/O Streams
Data Access with ADO.NET
Threading in C#
Next Steps & Q & A

Introduction

- Anil Joseph

- Over 20 years of Experience
- Technologies
  - C, C++
  - .NET
  - Java, Enterprise Java
  - React, Angular
  - Native Android, React Native, Xamarin
- Worked on numerous projects

Software

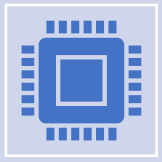
.NET SDK

VS.NET 2022

SQL Server

.NET Core: Powering Modern Software Development

.NET Core is a free, open-source, cross-platform framework for building modern, cloud-based, and IoT applications.



It is a popular choice for developers due to its flexibility and ability to run on various platforms.

.NET Core: Key Features

Cross-Platform: .NET Core allows developers to build applications that can run on Windows, macOS, and Linux.

Open Source: .NET Core is open-source, enabling collaboration and contribution from the community.

High Performance: .NET Core provides high performance and scalability, making it suitable for a wide range of applications.

Modular: .NET Core is designed with modularity in mind, allowing developers to include only the libraries needed for their applications.

Unified Platform: .NET Core unifies the .NET platforms, making it easier for developers to target different types of applications, such as web, desktop, mobile, gaming, and IoT.

Evolution of .NET Core

.NET Core 1.0 (June 2016):

- Cross-platform support and a lightweight, modular runtime.

.NET Core 2.0 and 2.1 (August 2017 - May 2018):

- Improved performance, better compatibility with existing .NET Framework libraries, and enhanced tools and APIs for developers.

.NET Core 3.0 (September 2019):

- Introduced Windows Presentation Foundation (WPF) and Windows Forms for desktop applications, along with enhanced support for cloud-native applications and machine learning scenarios.

Evolution of .NET Core

Unified Platform with .NET 5 (November 2020):

- .NET Core, .NET Framework, and Xamarin into a single platform.
- Improved consistency, performance, and developer experience.

.NET 6 and Beyond:

- Improved performance, and support for the latest technologies and industry trends.

Languages Supported in .NET Core

C#:

Most popular and widely used language in the .NET ecosystem.

Object-oriented, strongly-typed language with modern features such as async/await, LINQ, and lambda expressions.

F#:

F# is a functional-first programming language in the .NET family.

Suitable mathematical computations, data manipulation, and complex algorithm implementations.

Visual Basic .NET (VB.NET):

VB.NET is an object-oriented programming language and is an evolution of the classic Visual Basic language.

Other Languages

IronPython, IronRuby

Understanding CLR and CLS

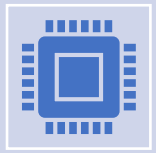
Common Language Runtime (CLR):

- CLR is the virtual machine that executes .NET applications.
- CLR provides various services such as memory management, exception handling, security enforcement, and garbage collection.

Common Language Specification (CLS):

- CLS defines a set of rules and guidelines that every language should follow when compiled to IL, ensuring interoperability between different .NET languages.
- Enables different .NET languages to interact seamlessly within the same application.

Benefits of CLR and CLS



Language Interoperability: CLR and CLS enable developers to use multiple languages within a single .NET application, leveraging the strengths of each language.

Component Reusability: Components developed in one language can be easily reused in projects developed using other .NET languages, promoting code reuse and collaboration.

Simplified Integration: CLR and CLS simplify integration tasks, allowing developers to build complex applications by combining modules written in different languages.

C#

C# is a versatile, modern, object-oriented programming language developed by Microsoft.

Its simple, readable, and expressive syntax, making it an excellent choice for both beginners and experienced developers.

C# Key Features

- **Type Safety**:
  - C# is a statically typed language, providing type safety at compile time, which helps catch errors before execution.
- **Object-Oriented**:
  - Supports object-oriented programming principles like encapsulation, inheritance, and polymorphism, enabling the creation of reusable and organized code.
- **Memory Management**:
  - Includes automatic memory management through garbage collection, reducing the risk of memory leaks and simplifying memory management tasks.
- **Asynchronous Programming**:
  - C# supports asynchronous programming through features like async/await, enabling efficient handling of I/O-bound operations.

C# Key Features

- **LINQ (Language Integrated Query)**:
  - Highlight LINQ, a powerful feature of C# that allows querying collections and databases using a uniform syntax, enhancing productivity and readability.
- **Exception Handling**:
  - Discuss C#'s robust exception handling mechanisms, aiding in the creation of stable and fault-tolerant applications.

.NET CLI

- .NET Command-Line Interface (CLI) is a powerful tool for managing .NET projects.
- Its cross-platform compatibility, allowing developers to use the same commands on Windows, macOS, and Linux.
- dotnet new - Project Creation:
  - Creating new projects from templates.
- dotnet build - Compilation:
  - Compiles the source code and its dependencies into executable binaries.
- dotnet run - Application Execution:
  - Executing the application

Data Types: Primitive

Integer Types:

- int: Represents 32-bit signed integers.
- long: Represents 64-bit signed integers.
- short: Represents 16-bit signed integers.
- byte: Represents 8-bit unsigned integers.

2. Floating-Point Types:

- float: Represents 32-bit single-precision floating-point numbers.
- double: Represents 64-bit double-precision floating-point numbers.

3. Character Type:

- char: Represents a single 16-bit Unicode character.

4. Boolean Type:

- bool: Represents a Boolean value (true or false).

Data Types: Reference

Objects:

object: Represents a base type for all other data types. Can store any value.

Strings:

string: Represents a sequence of characters and is used for text manipulation.

Classes and Objects

Objects represent real-world entities

- Example: Employee, Product, Order, Vehicle etc

A class in C# is a blueprint for creating objects.

- It defines a data structure along with methods to work on that data.

Classes are fundamental to object-oriented programming,

- Promotes modularity, reusability, and maintainability.

An object is an instance of a class.

- When a class is defined, no memory is allocated until an object of that class is created.
- Created using the keyword, invoking the class constructor.

Class Members

Data Members:	Represent the state of the object.
Properties:	Encapsulating the data-member of the object. Accessed using get and set methods.
Methods:	Functions defined within a class, performing actions and operations on the object's data.
Constructors:	Special methods called when an object is created, initializing the object's state.
Events:	Mechanism for communication between objects, allowing one object to notify other objects about an action.

Access Modifiers

Access modifiers in C# define the scope and visibility of class members.

They control the level of access that classes, methods, and other members have in C#.

Encapsulation: Helps in encapsulating the internal state of objects, ensuring data integrity.

Security: Restricts unauthorized access to sensitive data and methods.

Flexibility: Allows fine-grained control over the visibility of class members.

Types of Access Modifiers

Public:

- Members declared as public are accessible from any part of the program.
- Example: public class MyClass { }

Private:

- Members declared as private are accessible only within the same class.
- Example: private int _myPrivateField;

Protected:

- Members declared as protected are accessible within the same class and its derived classes.
- Example: protected void MyProtectedMethod() { }

Types of Access Modifiers

Internal:

- Members declared as internal are accessible within the same assembly (project).
- Example: internal class MyInternalClass { }

Protected Internal:

- Members declared as protected internal are accessible within the same assembly and its derived classes.
- Example: protected internal double MyProtectedInternalProperty { get; set; }

Static Members

Static members in C# belong to the type itself rather than to any specific instance of the type.

They are shared among all instances of the class and can be accessed without creating an instance of the class.

Static Members

Static Fields:

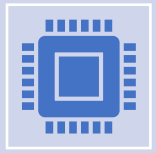
Variables declared with the static keyword, shared among all instances of the class.

Example: static int totalCount;

Static Properties:

Properties declared with the static keyword, providing access to static fields.

Example: public static int TotalCount { get; set; }



Static Methods:

Methods declared with the static keyword, can be called without creating an instance of the class.

Example: public static void PrintMessage() { Console.WriteLine("Hello, World!"); }

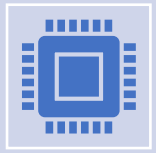
Static Members

Shared State:

Static members allow sharing state among all instances of a class.

Utility Methods:

Static methods can provide utility functions without the need for object instantiation.



Memory Efficiency:

Static members are allocated once in memory, saving memory space when compared to instance members.

Abstract Keyword

Used to declare an abstract class or abstract class members.

Abstract classes cannot be instantiated and are meant to be subclassed by concrete (non-abstract) classes.

Abstract methods must be implemented by derived classes.

Override Keyword

Used to override a virtual or abstract method in a derived class.

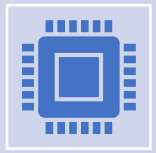
Provides a new implementation for a base class method in the derived class.

Virtual Keyword

Used to declare a method, property, or event that can be overridden in derived classes.

Allows base class methods to be customized in derived classes using the override keyword

Inheritance



Inheritance is a fundamental feature of object-oriented programming.

It allows a new class (derived class or subclass) to inherit properties and behavior (members) from an existing class (base class or superclass).

Enables code reuse, extensibility, and the creation of a hierarchy of classes.

Arrays

An array in C# is a collection of elements of the same data type.

Arrays allow you to store multiple values of the same type under a single variable name.

Elements in an array are accessed by an index, starting from 0 to the length of the array minus one.

Polymorphism

- It allows objects of different classes to be treated as objects of a common base class.
- Enables the implementation of methods in multiple derived classes with a common interface in the base class.
- Compile-time (Static) Polymorphism:
  - Achieved through method overloading and operator overloading.
  - Methods with the same name but different parameters can be called based on the context.

Polymorphism

- Run-time (Dynamic) Polymorphism:
  - Achieved through method overriding and interface implementation.
  - Allows the same method or property name to behave differently in derived classes.
  - Example: virtual void Draw() in base class overridden in derived classes.