.NET and C# - Clean Study Notes (ASCII Safe)

Grammar corrected and enriched. All special hyphens and dashes removed to prevent black boxes.

Generated: 2025-07-29 14:59

Overview - .NET

.NET is a free, cross-platform, open-source developer platform for building many types of applications (web, cloud, desktop, mobile, IoT, services). It consists of the following parts:

Runtime Environments

- .NET (current): unified, cross-platform runtime for Windows, Linux, and macOS.
- .NET Framework: the original Windows-only runtime for desktop and older web apps.
- Mono: open-source implementation used historically on non-Windows platforms.
- Languages: first-class languages include C#, F#, and Visual Basic.

Class Libraries

- Pre-built libraries covering file I/O, networking, data access, web development, cryptography, and more.
- Base Class Library (BCL): the core API set shared across .NET implementations.

Common Language Runtime (CLR)

- Executes .NET programs and provides memory management (garbage collection), type safety, exception handling, and JIT compilation.

Frameworks and Tools

- ASP.NET: web apps and APIs.
- Entity Framework: ORM for data access.
- WinForms / WPF: Windows desktop UI.
- Xamarin / .NET for iOS & Android (MAUI successor): mobile application development.
- .NET Standard: specification of APIs common to .NET implementations.

C# - Quick Overview

- Modern, object-oriented, type-safe language with exceptions, generics, LINQ, async/await, pattern matching, and spans.
- Used for desktop, web, cloud, mobile, and games within the .NET ecosystem.
- Great tooling with Visual Studio and VS Code.

Classes - Key Points

- Classes define state and behavior (fields, properties, methods, events).
- Classes are reference types; instances live on the managed heap.
- All types ultimately inherit from System.Object.
- Default access for a top-level class is internal. Default access for members is private.

Types of Classes

Abstract

- Cannot be instantiated; must be inherited.

- May contain abstract members (no implementation) and non-abstract members.
- C# supports single inheritance for classes (you can implement multiple interfaces).
- Non-abstract derived classes must implement all abstract members.

Partial

- One class split across multiple files; parts are combined at compile time.
- If any part is sealed, the entire class becomes sealed.
- Base class and access modifiers apply to the combined type.

Static

- Cannot be instantiated or inherited; only static members are allowed.

Sealed

- Prevents inheritance; useful for locking down behavior and enabling de-virtualization by the JIT.

Access Modifiers

- public: accessible from anywhere.
- private: accessible only within the containing type.
- protected: accessible within the containing type and derived types.
- internal: accessible within the same assembly.
- protected internal: protected OR internal.
- private protected: protected AND internal (derived types in the same assembly).

Virtual, Override, and Sealed

- virtual: allows derived classes to override behavior.
- override: provides a new implementation of a virtual member.
- sealed override: stops further overriding down the inheritance chain.

```
public class Animal
{
    public virtual string Speak() => "???";
}

public class Dog : Animal
{
    public sealed override string Speak() => "Woof!";
}
```

Garbage Collection (GC)

- CLR garbage collector reclaims memory for unreachable managed objects.
- Runs on its own threads; handles generations, compaction, and optional finalization.
- Unmanaged resources (file handles, sockets, native memory) are not freed by GC and must be released explicitly.

IDisposable and using

- Implement IDisposable when you own unmanaged resources or hold other IDisposable objects.

- Call Dispose() to deterministically release resources; prefer using / await using.

```
public sealed class FileWriter : IDisposable
{
    private readonly FileStream _stream;
    private bool _disposed;

    public FileWriter(string path) => _stream = File.Create(path);

    public void Write(byte[] data)
    {
        if (_disposed) throw new ObjectDisposedException(nameof(FileWriter));
        _stream.Write(data, 0, data.Length);
    }

    public void Dispose()
    {
        if (_disposed) return;
        _stream.Dispose();
        _disposed = true;
    }
}

using var writer = new FileWriter("out.bin");
writer.Write(new byte[] { 1, 2, 3 });
```

Finalizers

- Finalizers (~TypeName) run before GC reclaims an object, as a last resort to release unmanaged resources.
- Cannot be called directly; non-deterministic and add overhead.
- Prefer SafeHandle plus Dispose; use finalizers only as a safety net.

```
class Car
{
    ~Car()
    {
        // cleanup for unmanaged resources
    }
}
```

Value Types vs Reference Types

- Value types (structs, enums) are copied by value; they may live on the stack, inside objects, or in arrays.
- Reference types (classes, arrays, strings, delegates) are allocated on the heap and accessed via references; they can be null.
- Boxing converts a value type to object (allocates); unboxing extracts the value.

Parameter Passing - value, ref, in, out

- By value (default): the callee receives a copy (for reference types: a copy of the reference).
- ref: pass by reference; the callee can read and write the caller's variable.
- in: pass by reference, read-only.
- out: pass by reference; the callee must assign before returning.

```
void ParseAndBump(in int x, out int y)
{
```

```
    y = x + 1; // must assign
}

int a = 10;
ParseAndBump(a, out var b); // a unchanged; b == 11
```

Collections and Generics

- Arrays: fixed size, contiguous memory, O(1) indexing.
- List: dynamic array-backed list; fast indexing and append.
- Dictionary: hash table; O(1) average lookup/insert/delete.
- ConcurrentDictionary: thread-safe dictionary for multi-threaded scenarios.
- Prefer generic collections in System.Collections.Generic. Avoid non-generic collections to prevent boxing and to keep type-safety.

Under the Hood

- Arrays: contiguous block; elements are stored sequentially.
- List: wraps an internal T[]; fields like int _size and int _version; growth re-allocates a larger array and copies elements.
- Dictionary: hashing places entries into buckets; a good hash distribution is important for performance.

Performance Notes

- Array index lookup: O(1).
- List: O(1) by index; O(n) for search by value without a key.
- Dictionary: O(1) average for lookup/insert/delete.
- Minimize allocations. Use structs only when small and cheap to copy.

Threading, Task, and async/await

- Threading lets you run multiple flows of execution concurrently or in parallel.
- Use low-level System.Threading APIs sparingly; prefer higher-level abstractions such as ThreadPool, Task, and async I/O.

Task vs Thread

- Thread: OS thread of execution; manual lifecycle and synchronization.
- Task: higher-level abstraction for units of work; supports composition (continuations, WhenAll), cancellation, and exception propagation.

async/await - how it works

- The compiler transforms an async method into a state machine.
- If an awaited task is not complete, control returns to the caller; when it completes, the continuation resumes.
- Enables non-blocking, scalable I/O.

```
public async Task<string> GetDataAsync()
{
    string data = await SomeAsyncMethod();
    return data;
```

```
}
```

Delegates and Events

Delegates

- A delegate is a type-safe function pointer describing a method signature.
- Only methods with matching signature can be assigned to the delegate.

```
public delegate string IntToString(int i);

public static string IntToBinaryString(int i) => Convert.ToString(i, 2);
public static string IntToHexString(int i)    => Convert.ToString(i, 16);

public static void PrintIntAsString(IntToString convert, int i) =>
    Console.WriteLine(convert(i));
```

Events

- An event exposes a delegate for subscribers but only the declaring type can raise it.
- Handlers must match the event's delegate signature (for example, EventHandler).

```
class ProgramTwo
{
    static void Main()
    {
        var c = new Counter();
        c.ThresholdReached += C_ThresholdReached;
    }

    static void C_ThresholdReached(object sender, EventArgs e) =>
        Console.WriteLine("The threshold was reached.");
}
```

Handling and Throwing Exceptions

- System.Exception is the base class for all exceptions.
- Use Message, StackTrace, and InnerException for diagnostics.
- Prefer existing framework exceptions; use finally to run cleanup code that must always execute.

Connecting to a Data Source (ADO.NET)

- Use a connection object (for example, SqlConnection) with a connection string to connect to the data source.
- Typical flow: open connection -> create command -> execute -> read results -> dispose.

Mutability

Mutable vs Immutable

- Mutable examples: StringBuilder, most user-defined classes, Dictionary.
- Immutable examples: string; for immutable collections use System.Collections.Immutable.

```
string s1 = "Hello ";
string s2 = s1;
s1 += "World";
Console.WriteLine(s2); // Hello
```

The using Keyword

- Using directive: imports a namespace.
- Using statement: scopes a disposable object and calls Dispose() automatically at the end of the scope.

```
// Directive
using System.IO;

// Statement
using var stream = File.OpenRead("file.txt");
// stream is disposed at end of scope
```

Object Lifetimes in ASP.NET Core DI

Lifetime	Meaning
Transient	New instance each time it is requested.
Scoped	One instance per request (web scope).
Singleton	Single instance for the lifetime of the application.

Managed vs Unmanaged Code

- Managed code runs under the CLR (JIT compiled; GC handles memory).
- Unmanaged code runs outside the CLR (for example, native C or C++ applications) and manages memory manually.

CTS and CLS

Common Type System (CTS)

- Defines how types are declared, used, and managed across languages so they interoperate.

Common Language Specification (CLS)

- A set of rules for library authors to ensure cross-language compatibility.
- CLS-compliant libraries can be consumed from any CLS-compliant language.

Casting

Implicit Casting

```
int i = 10;
double d = i; // widening conversion
```

Explicit Casting

```
double d1 = 100.23;
int y = (int)d1; // truncates: y == 100
```

Explicit casts may lose information; use them intentionally.

Array vs ArrayList

- Array: fixed length, strongly typed, best performance.
- ArrayList: non-generic, stores object, causes boxing for value types. Prefer List.

The out Keyword

Both the method definition and the call must use out. The callee must assign the variable before returning.

```
int n;
OutArgExample(out n);
Console.WriteLine(n); // 44

void OutArgExample(out int number)
{
    number = 44;
}

// Inline out variable
if (int.TryParse("1640", out int value))
    Console.WriteLine($"Converted to {value}");
```

Assemblies and DLLs

- An assembly is the basic deployment unit (.exe or .dll) containing IL and metadata.
- A DLL (Dynamic Link Library) is a library assembly that other programs can reference.

End of ASCII-safe notes.