# C# — Topic 01: Environment & Project Basics

Filename: CSharp-Topic-01-Environment-and-Project-Basics.pdf

**Objective**: Understand the .NET SDK and CLI, SDK-style project structure, and the build/run/publish lifecycle for console apps.

**Difficulty**: 1/5 **Estimated Time**: 1.5–2 hours **Prereqs**: none **Dependencies**: —

## 1) .NET SDK & CLI

- The .NET SDK includes the compiler, CLI (dotnet), and build tools. It is used on all platforms.
- The dotnet CLI has two broad roles: (a) developer workflow (create/build/run/publish), and (b) executing .NET applications.
- Templates (e.g., console) are created with the CLI and use SDK-style projects by default.

Example (inspect your installation):

```
dotnet --info dotnet --list-sdks
```

## 2) SDK-Style Project Structure

- SDK-style projects centralize build logic via the Sdk attribute and MSBuild properties.
- The target framework (TFM) selects APIs/runtime (e.g., net8.0, net9.0).
- Nullable and implicit usings are commonly set per project to guide code quality and ergonomics.

Minimal console project file:

```
<Project Sdk="Microsoft.NET.Sdk"> <PropertyGroup>
<TargetFramework>net8.0</TargetFramework> <Nullable>enable</Nullable>
<ImplicitUsings>enable</ImplicitUsings> </PropertyGroup> </Project>
```

## 3) Program Structure & Entry Point

- Top-level statements remove the need for an explicit Program.Main; the compiler generates the entry point.
- You can always switch to an explicit Main if you need custom signatures, early return codes, or clarity.

Top-level statements (Program.cs):

```
Console.WriteLine("Hello, .NET!");
```

Equivalent explicit entry point:

```
namespace MyApp; public static class Program { public static int Main(string[] args)
{ Console.WriteLine("Hello, .NET!"); return 0; } }
```

## 4) Build, Run, Publish — Lifecycle

- build: compiles the project for developer use (bin/Debug|Release/TFM). It does not produce a deployment layout.
- run: builds if necessary and then executes the app (useful during development).

- publish: produces a deployment-ready output folder. Use this for distribution or deployment.

Example commands:

```
dotnet build -c Release dotnet run -- Hello dotnet publish -c Release -r linux-x64
```

## 5) App Host & Deployment Modes

- Framework-dependent deployment: uses the shared .NET runtime on the target machine; publish produces a platform-specific app host plus your binaries.
- Self-contained deployment: bundles the .NET runtime; larger output but no shared runtime required.
- Runtime Identifier (RID) selects the target platform (e.g., linux-x64, win-x64); match the RID to your deployment target.

Running the app host on Linux (framework-dependent):

```
./bin/Debug/net8.0/MyApp # after build ./bin/Release/net8.0/linux-x64/publish/MyApp #
after publish
```

## 6) Output Layout & Configuration

- bin/: build outputs grouped by configuration (Debug/Release) and TFM; publish/ contains deployment assets.
- obj/: intermediate build artifacts (do not depend on these at runtime).
- Configuration matters for optimizations and diagnostics; Release is recommended for publish.

## 7) Common MSBuild Properties (Theory)

```
<PropertyGroup> <TargetFramework>net8.0</TargetFramework> <Nullable>enable</Nullable>
<ImplicitUsings>enable</ImplicitUsings>
<TreatWarningsAsErrors>false</TreatWarningsAsErrors> </PropertyGroup>
```

## Summary

- Use the .NET SDK and CLI to manage the full lifecycle of console apps across platforms.
- Prefer SDK-style projects with clear properties; choose the TFM that matches your deployment/runtime goals.
- Use top-level statements for simplicity or an explicit Main when you need control over arguments and exit codes.
- Publish for deployment; pick framework-dependent vs self-contained based on runtime availability and size constraints.