SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Object-Oriented Programming with C#

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

Object-oriented programming is the design language of the .NET world: every framework you will use, from ASP.NET Core to Entity Framework, is built from classes, interfaces, and inheritance, and it assumes you can read and write that language fluently. Many developers can use the syntax without ever being taught the design thinking behind it, and it shows: rigid class hierarchies, god objects, and code that fights every change. The syntax is easy; knowing what deserves to be a class, and what that class should hide, is the skill.

This is a hands-on, practitioner course. It builds object-oriented C# in dependency order: first classes and encapsulation, because everything else refines them, then inheritance and polymorphism, then interfaces and abstraction, and finally the design principles that tell you when to use each. Rather than survey the entire C# language, it goes deep on the object-oriented core and the judgment to use it well, developing one domain model across the course. Every module ends with a lab and builds on the one before.

Who Should Attend

- Developers who know basic programming and want to learn C# and OO design properly
- Developers coming to C# from Python, JavaScript, or another language
- C# developers who learned on the job and want the design foundations they skipped

Learners who have never programmed should take *Introduction to Programming* first.

Prerequisites

- Working proficiency in at least one programming language
- Understanding of variables, control flow, functions, and collections
- No prior C# or .NET experience required

What You Will Learn

- Write idiomatic C#: types, properties, methods, and the conventions of the .NET world
- Design classes that encapsulate state and expose honest, minimal interfaces
- Apply inheritance and polymorphism where they help, and recognize where they hurt
- Use interfaces and abstraction to decouple code and make it testable
- Judge designs with core principles: single responsibility, composition over inheritance, dependency inversion
- Build a small, well-structured object-oriented application in C#

Course Outline

Day one: objects done right

- C# and .NET Bearings

  - The .NET ecosystem: runtime, SDK, and project structure
  - C# essentials for experienced programmers: types, syntax, and conventions
  - Value types versus reference types, and why it matters constantly
  - Lab: set up the course project and port a small program you already understand into C#
- Classes and Encapsulation
  - Classes, objects, constructors, and properties
  - Encapsulation: hiding state, exposing behavior, and why getters everywhere is a smell
  - Members, access modifiers, and designing a class's public face
  - Lab: build the first classes of the course domain model with real encapsulation
- Inheritance and Polymorphism
  - Base classes, derived classes, and virtual and override
  - Polymorphism: one call, many behaviors
  - Abstract classes, and the honest costs of deep hierarchies
  - Lab: extend the domain model with an inheritance hierarchy and make polymorphism do the work

Day two: abstraction, design judgment, and a working application

- Interfaces and Abstraction
  - Interfaces as contracts: depending on what, not how
  - Interfaces versus abstract classes: choosing deliberately
  - How abstraction enables substitution and testing
  - Lab: introduce interfaces into the domain model and swap an implementation without touching its consumers
- Design Principles in Practice
  - Single responsibility and small, focused classes
  - Composition over inheritance: the fix for most hierarchy pain
  - Dependency inversion, with a first look at dependency injection in .NET
  - Lab: refactor a tangled design using the principles, and defend each change
- Building a Real OO Application
  - Collections and generics: List<T>, Dictionary<TKey, TValue>, and writing generic code
  - Exceptions: failing loudly, clearly, and in the right place
  - Assembling the pieces: a small layered application built from your domain model
  - Lab: complete a working console application on the course domain model, end to end

Extended Version

The three-day version keeps the same gradient and adds depth and modern C# range:

- Modern C# idioms: records, pattern matching, nullable reference types, and LINQ fundamentals
- Unit testing object-oriented code, and how good design makes testing easy
- A first look at design patterns as named applications of the course's principles
- A capstone: design and build a complete small application from a written brief, then walk through the design decisions