ENGINEER-TO-ARCHITECT AND DURABLE THINKING SKILLS

Design Patterns in Practice

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

Overview

Design patterns are the shared vocabulary of software design: named, proven solutions to problems that recur in every non-trivial codebase. The hard part is not memorizing the catalog. It is recognizing the problem a pattern solves when you meet it in real code, and having the judgment to know when a pattern clarifies a design and when it just adds ceremony.

This is a hands-on, practitioner course. Rather than march through every pattern in the catalog, it goes deep on the patterns you will actually meet in working code, in an order that builds: first the mental model of what a pattern is and how to read one, then creational, structural, and behavioral patterns applied to real code, and finally the judgment to know when a pattern is the wrong answer. Less content, taught more effectively: the patterns we skip are easy to pick up once the core set is solid. Every module ends with a lab, and each module builds on the one before.

Who Should Attend

- Developers who want to design cleaner, more maintainable object-oriented code
- Senior developers who know some patterns by name but want practiced judgment about using them
- Technical leads who review designs and want a shared vocabulary for their teams

Learners who are new to object-oriented programming should take *Object-Oriented Programming with C#* first.

Prerequisites

- Working proficiency in an object-oriented language such as C#, Java, or Python
- Comfort with classes, interfaces, and inheritance
- Experience working in a codebase larger than a toy project

What You Will Learn

- Explain what a design pattern is, the problem-context-tradeoff structure behind every pattern, and why the vocabulary matters
- Recognize patterns already at work in the frameworks and codebases you use every day
- Apply the core creational patterns to control how objects get built
- Apply the core structural patterns to compose objects into flexible designs
- Apply the core behavioral patterns to manage varying behavior and communication between objects
- Judge when a pattern earns its complexity and when simpler code is the better design

Course Outline

Day one: reading and recognizing patterns

- Pattern Thinking
  - What a pattern is: a named solution, its context, and its tradeoffs

  - Patterns as a design vocabulary for teams and code reviews
  - The underlying principles: program to interfaces, favor composition over inheritance
  - Lab: find and name three patterns already at work in a familiar framework or codebase
- Creational Patterns
  - Factory Method and Abstract Factory: controlling what gets created
  - Builder: constructing complex objects step by step
  - Singleton: what it promises, and why it is the most misused pattern in the catalog
  - Lab: refactor tangled construction code using a factory and a builder
- Structural Patterns
  - Adapter and Facade: making incompatible or complicated things usable
  - Decorator: adding behavior without inheritance
  - Composite: treating parts and wholes uniformly
  - Lab: wrap a messy third-party interface with an adapter and a facade

Day two: behavior and judgment

- Behavioral Patterns
  - Strategy: swapping algorithms without conditionals
  - Observer: decoupling things that need to react to each other
  - Command and Template Method: encapsulating actions and varying steps
  - Lab: replace a growing conditional with Strategy, then add an Observer to decouple a notification
- Patterns in Modern Codebases
  - Patterns your language already gives you: delegates, lambdas, and built-in features that replace boilerplate
  - Patterns baked into frameworks: dependency injection, middleware pipelines, event systems
  - Refactoring toward a pattern instead of designing one in up front
  - Lab: refactor a working but rigid module toward a pattern, driven by a new requirement
- The Judgment to Say No
  - Overengineering: pattern-itis and speculative generality
  - Code smells that genuinely call for a pattern versus complexity for its own sake
  - Reviewing a design: asking what problem each pattern is earning its keep against
  - Lab: review a pattern-heavy design, identify what to keep and what to simplify, and defend the calls

Extended Version

The three-day version keeps the same gradient and adds depth and practice:

- More behavioral patterns in depth: State, Chain of Responsibility, and Mediator
- Patterns in concurrent and asynchronous code
- Anti-patterns and recovering from pattern misuse in legacy code
- A capstone: redesign a realistic module end to end, applying and defending pattern choices in a team review