Curso ATDEV320 - Design Patterns e Boas Práticas em .NET

Duração: 35h

Introdução

Como se torna um programador melhor? Uma das melhores formas é aprender design patterns. Os design patterns oferecem soluções reutilizáveis para problemas recorrentes de desenho de software. Nesta formação identificamos problemas de desenho e exploramos como resolvê-los com o design pattern mais adequado. Abordamos padrões creational, behavioral e structural e, claro, exploramos as opções de implementação em .NET. Todos os exemplos e laboratórios usam a versão LTS mais recente do .NET e do Visual Studio, mas funcionam também com qualquer versão recente.

Público-alvo

Este curso destina-se a programadores experientes, familiarizados com C# e com experiência prática de .NET.

No final do curso

Após a conclusão deste curso, os formandos deverão ser capazes de:
• Aplicar os princípios S.O.L.I.D. no desenho orientado a objetos.
• Reconhecer e aplicar os design patterns criacionais, estruturais e comportamentais.
• Utilizar value objects e os patterns Specification, Repository e Result no acesso a dados.
• Construir interfaces flexíveis com padrões da família Model-View.
• Desenvolver uma biblioteca reutilizável assente em design patterns.

Pré-requisitos

Experiência prática de .NET e familiaridade com C#.

Exames de certificação

(não existem exames)

Conteúdo programático em detalhe

Melhorar o desenho orientado a objetos com os princípios S.O.L.I.D.
• S — responsabilidade única
• O — aberto/fechado
• L — substituição de Liskov
• I — segregação de interfaces
• D — inversão de dependências
Introdução aos patterns
• O que é um pattern?
• O Gang of Four: Erich Gamma, Ralph Johnson, Richard Helm e John Vlissides.
• Diferentes tipos de design pattern: creational, structural e behavioral.
• Patterns por todo o lado: a diferença entre patterns de implementação, de desenho e de arquitetura.
• Quando aplicar patterns — e quando não os aplicar

www.actualtraining.pt
Tel: +351 217 158 018

• Alguns anti-patterns, como o Loosey-Goosey

Singleton pattern

• O que é um Singleton?
• Opções de implementação do Singleton em .NET
• Utilizar dependency injection
• Por que selar classes é boa ideia
• LAB: construir um Singleton em .NET

Reutilização de código com Template Method

• Template Method — adiar partes exatas de um algoritmo para classes derivadas, delegates e outros
• LAB: construir um Template Method para um jogo de tabuleiro

Criação abstrata de objetos com factories

• Factory Method — delegar e esconder numa factory a criação de objetos
• Abstract Factory — abstrair numa factory a criação de famílias de objetos
• LAB: implementar uma Abstract Factory

Os patterns Strategy e Chain-of-Responsibility

• Strategy — o Template Method sem a herança incómoda
• LAB: ordenar objetos com strategies
• Chain of Responsibility — a strategy que percorre uma cadeia de strategies
• LAB: usar uma Chain of Responsibility para implementar um Builder

Collection patterns

• Iterator — uma forma genérica de percorrer coleções; o yield é seu amigo; iterar uma coleção de forma assíncrona
• Considerações de desempenho na implementação de IEnumerable
• Utilizar frozen collections

Behavioral pattern: State e máquinas de estados

• Reconhecer o problema: lógica condicional espalhada por uma classe
• State — encapsular comportamento em objetos de estado e tornar as transições explícitas
• Modelar fluxos de trabalho como máquinas de estados finitos
• Usar a biblioteca Stateless: definir estados, triggers e guards em .NET
• Visualizar máquinas de estados para documentação e depuração
• LAB: implementar um leitor multimédia (reproduzir/pausa/parar) com o State pattern e a biblioteca Stateless

Structural pattern: Flyweight

• Reconhecer o problema: milhares de objetos quase idênticos a consumir memória em excesso
• Flyweight — separar o estado intrínseco (partilhado) do estado extrínseco (contextual)
• Implementar uma factory de Flyweight com cache em .NET
• Como o .NET já aplica o Flyweight: string interning e cache de inteiros pequenos
• Compromissos: quando o Flyweight ajuda — e quando acrescenta complexidade desnecessária

Structural pattern: Composite

• Composite — construir hierarquias parte-todo com uma interface uniforme
• Nós folha vs. nós compostos: quando acrescentar a gestão de filhos à interface do componente
• Composite em .NET: expression trees e árvores de controlos de interface

Behavioral pattern: Interpreter

• Interpreter — representar as regras de uma gramática como hierarquia de classes
• Construir e avaliar uma árvore de sintaxe abstrata (AST)
• Como as expression trees do LINQ usam o Interpreter — e como construir a sua

• LAB: construir um avaliador simples de expressões aritméticas com o Interpreter

Creational pattern: Builder

• Builder — construção passo a passo de objetos complexos
• APIs de builder fluentes: encadear métodos para uma construção legível e explícita na intenção
• Builder em .NET: WebApplicationBuilder, IHostBuilder e builders de dados de teste
• Classes director: encapsular sequências de construção comuns
• LAB: analisar expressões com o Builder pattern

Behavioral pattern: Visitor

• Visitor — separar os algoritmos da estrutura de objetos sobre a qual atuam
• Double dispatch: como o Accept e o Visit funcionam em conjunto
• Construir um pretty-printer com o Visitor clássico
• Alternativa moderna: pattern matching em C# com switch expressions
• Usar dynamic dispatch para evitar o código repetitivo do Accept
• Compromissos: Visitor clássico vs. pattern matching — quando escolher cada um
• LAB: implementar um Visitor que gera vários formatos de saída a partir da mesma hierarquia de objetos

Behavioral pattern: Observer

• Observer — desacoplar quem publica de quem subscreve
• Eventos e delegates em .NET: a implementação do Observer que vem de origem
• As interfaces IObservable<T> e IObserver<T> como alternativa assente em push
• LAB: construir um teletipo de cotações com o Observer

Behavioral pattern: Mediator

• Mediator — um centro único que substitui as referências diretas entre objetos
• Observer vs. Mediator: quando usar cada um
• Implementar corretamente o INotifyPropertyChanged — e evitar o anti-padrão das strings soltas
• Usar nameof() e source generators para tornar as notificações de propriedade seguras à refatoração
• A biblioteca MediatR: aplicar o Mediator numa arquitetura de serviços e handlers
• LAB: refatorar um formulário fortemente acoplado para um desenho assente no Mediator

Structural pattern: Adapter

• Adapter — ligar interfaces incompatíveis sem tocar em nenhuma das classes
• Object adapter vs. class adapter: composição vs. herança
• Adaptar bibliotecas de terceiros às interfaces do seu domínio
• Adapters bidirecionais: envolver os dois sentidos
• LAB: construir um adapter com expressions e o DLR

Structural pattern: Decorator

• Decorator — acrescentar comportamento de forma transparente, sem alterar quem chama
• Empilhar decorators: compor preocupações transversais (logging, caching, validação)
• Decorator em .NET: Stream, middleware do ASP.NET Core e decoração via DI
• Registar decorators num contentor de DI
• LAB: acrescentar operações de ordenação com decorators

Structural pattern: Proxy

• Proxy — controlar o acesso ao objeto real
• Virtual proxy: adiar a criação dispendiosa de um objeto até ser necessária
• Protection proxy: impor autorização antes de delegar as chamadas
• Remote proxy: fazer com que uma chamada de rede pareça uma chamada local

• Proxy vs. Decorator: a mesma estrutura, intenções diferentes

Esconder detalhes de implementação com Facade

• Facade — esconder de quem chama a complexidade dos subsistemas

Utilização de value objects

• Evitar o anti-padrão da obsessão por primitivos
• Boas práticas na construção de value objects
• LAB: construir um value object de endereço de correio

Acesso a dados com os patterns Specification e Repository

• Repository — abstrair o acesso a dados
• LAB: implementar um Repository com Entity Framework Core
• Specification — encapsular queries complexas
• Usar o Specification pattern com Entity Framework Core
• LAB: implementar uma Specification para um Repository

Utilização do Result pattern

• As exceções são dispendiosas
• Utilizar o Result pattern
• Melhorar o Result pattern com sintaxe LINQ
• LAB: usar o Result pattern numa aplicação MVC

Construir interfaces flexíveis com Model-View-Whatever

• Model-View-Controller: um padrão antigo de volta à moda
• O padrão MVC do ASP.NET
• MVVM em Blazor — o MVW a tirar partido de capacidades poderosas de data binding
• Command: encapsular comportamento em objetos
• Implementar commands com closures

Design patterns aplicados: desenvolver a sua própria biblioteca reutilizável

• Acrescentar o Command pattern do GoF ao MVVM
• Usar interfaces para ganhar flexibilidade
• Construir objetos de comando — estender a interface ICommand
• Acrescentar funcionalidade de anular e refazer ao Command pattern
• Usar uma classe CommandManager
• Desafio: encaixar os nossos commands no MVVM sem grandes alterações
• Implementar anular/refazer com o Memento pattern
• Decidir se vale a pena acrescentar o Prototype pattern
• Ideias para prosseguir com o Command pattern
• LAB: implementar a lógica de anular/refazer como biblioteca reutilizável