Módulo 1

**Padrões de projeto**

**Resumo: Padrões de Projeto**

Os **padrões de projeto** são soluções reutilizáveis para problemas comuns no desenvolvimento de software. Eles melhoram a **qualidade do código**, facilitam a **manutenção** e criam um **vocabulário comum** entre os desenvolvedores, o que melhora a comunicação entre equipes.

**Por que são importantes?**

* Evitam que cada equipe "reinvente a roda".
* Ajudam a **organizar melhor o código**.
* Tornam mais fácil trabalhar em sistemas já existentes.
* Reduzem o esforço e o tempo no desenvolvimento.
* Servem como **boas práticas testadas** ao longo do tempo.

**Analogia com o xadrez**

Assim como enxadristas experientes estudam jogadas conhecidas (como “aberturas”), **desenvolvedores experientes usam padrões** para resolver problemas já enfrentados por outros.  
Isso acelera decisões e melhora os resultados.

**Gang of Four (GoF)**

Em 1994, quatro engenheiros de software — **Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides** — lançaram o livro:

**"Design Patterns: Elements of Reusable Object-Oriented Software"**

Nele, descreveram **23 padrões de projeto** que ficaram conhecidos como **padrões GoF**, amplamente utilizados até hoje.

**Exemplos de padrões famosos**

* **Singleton** – Garante que uma classe tenha apenas uma instância.
* **Observer** – Um objeto notifica outros quando seu estado muda.
* **Factory Method** – Cria objetos sem precisar especificar a classe exata.

**Conclusão**

Conhecer e aplicar padrões de projeto é essencial para se tornar um desenvolvedor melhor. Assim como no xadrez, estudar boas jogadas permite tomar **decisões mais rápidas, mais inteligentes** e com menos erros.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

**Conhecendo o padrão de projeto**

**Resumo: Conhecendo o Padrão de Projeto**

Um **padrão de projeto (design pattern)** é uma **solução nomeada e reutilizável** para um problema recorrente no desenvolvimento de software. Ele descreve **como estruturar a solução**, geralmente por meio de módulos, classes, interfaces e interações.

**Características principais dos padrões**

Cada padrão é composto por quatro elementos fundamentais:

1. **Nome** – Permite que desenvolvedores se comuniquem de forma rápida e objetiva. Ex: "Use o padrão *Observer*".
2. **Problema** – Explica o contexto, quando usar, e o tipo de desafio que o padrão resolve.
3. **Solução** – Define a estrutura conceitual e como os elementos interagem. A implementação é feita pelo desenvolvedor.
4. **Consequências** – Avalia os impactos do padrão, como **flexibilidade**, **complexidade** e **performance**.

**Vantagens do uso de padrões**

* **Produtividade**: economizam tempo com soluções já validadas.
* **Reutilização**: facilitam alterações sem grandes quebras, evitando herança rígida.
* **Comunicação**: criam um **vocabulário comum** entre desenvolvedores.
* **Facilidade**: permitem reutilizar **boas práticas** em problemas semelhantes.

**Pontos de atenção / Desvantagens**

* Há **muitos padrões** disponíveis, o que exige estudo e prática.
* **Escolher o padrão certo** para um problema pode ser difícil.
* Padrões **devem ser adaptados** conforme o contexto, e isso exige experiência.
* Iniciantes podem **usar padrões em excesso** ou de forma inadequada.
* Nem todos os padrões são unânimes (exemplo: o **Singleton** é polêmico).

**Conclusão**

Padrões de projeto não são receitas prontas, mas **formas comprovadas de pensar e estruturar soluções**. Aprender e aplicar esses padrões com sabedoria pode elevar a qualidade dos sistemas desenvolvidos e facilitar a colaboração entre profissionais.

**Principais padrões em linhas gerais**

**Resumo: Principais Padrões de Projeto GoF**

Os padrões de projeto da chamada **“Gang of Four” (GoF)** são amplamente utilizados no desenvolvimento de software orientado a objetos. Eles ajudam a lidar com **criação de objetos, estruturação de sistemas e definição de comportamentos**, promovendo reutilização, flexibilidade e organização no código.

**🔹 Categorias dos Padrões GoF**

**1. Padrões Criacionais**

Focam **na criação de objetos** de forma flexível e independente da implementação concreta.

* **Abstract Factory** – Cria famílias de objetos relacionados.
* **Builder** – Separa a construção de um objeto complexo da sua representação.
* **Factory Method** – Define uma interface para criação, mas deixa as subclasses decidirem qual classe instanciar.
* **Prototype** – Cria objetos a partir da clonagem de um protótipo existente.
* **Singleton** – Garante que uma classe tenha **uma única instância** e fornece um ponto de acesso global.

**2. Padrões Estruturais**

Lidam com a **composição de classes e objetos** para formar estruturas mais complexas.

* **Adapter** – Converte a interface de uma classe em outra esperada.
* **Bridge** – Separa a abstração da sua implementação.
* **Composite** – Cria estruturas hierárquicas de objetos (ex: árvore de elementos).
* **Decorator** – Adiciona funcionalidades a objetos dinamicamente.
* **Facade** – Fornece uma interface simplificada para um subsistema.
* **Flyweight** – Compartilha objetos para reduzir o uso de memória.
* **Proxy** – Controla o acesso a um objeto, podendo adicionar lógica extra.

**3. Padrões Comportamentais**

Tratam da **comunicação entre objetos** e da **distribuição de responsabilidades**.

* **Chain of Responsibility** – Passa uma solicitação por uma cadeia de objetos.
* **Command** – Encapsula uma solicitação como um objeto.
* **Interpreter** – Avalia expressões de uma linguagem.
* **Iterator** – Permite percorrer os elementos de uma coleção sem expor sua estrutura.
* **Mediator** – Controla a comunicação entre objetos para reduzir o acoplamento.
* **Memento** – Permite restaurar um estado anterior de um objeto.
* **Observer** – Permite que objetos sejam notificados sobre mudanças em outro objeto.
* **State** – Altera o comportamento de um objeto conforme seu estado interno.
* **Strategy** – Define uma família de algoritmos intercambiáveis.
* **Template Method** – Define o esqueleto de um algoritmo em uma classe base.
* **Visitor** – Permite adicionar operações a objetos sem modificar suas classes.

**✅ Conclusão**

Você **não precisa memorizar todos os padrões**, mas deve saber **quando e onde encontrá-los**, além de entender **quais problemas eles resolvem**. Assim, quando enfrentar desafios semelhantes, poderá aproveitar **soluções testadas e aprovadas por outros desenvolvedores**.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Módulo 2

**Especialista**