02

O que é injeção de dependência e por que ela é necessária?

Vamos dar uma olhada no código do PedidoController:

```
    public class PedidoController : Controller
    {
        private readonly IProdutoRepository produtoRepository;

        public PedidoController(IProdutoRepository produtoRepository)
        {
            this.produtoRepository = produtoRepository;
        }
...
```

Note que esse construtor está exigindo um parâmetro: `public PedidoController(IProdutoRepository produtoRepository)`. Esse parâmetro é uma **dependência**, isto é, sem ele o controller não vai funcionar, pois precisamos do repositório de produtos para podermos consultar os produtos do e-commerce.

Mas como esse parâmetro é injetado? Note que, **em nenhum momento**, nós instanciamos um controller diretamente, como por exemplo `PedidoController meuController = new PedidoController()`, certo? Quando chamamos uma url para acessar action através do navegador, o controller é instanciado pelo próprio ASP.NET Core, por baixo dos panos. O que fazemos na injeção de dependência é informar o ASP.NET Core **sobre tudo o que a instância do controller precisa para ser criada**. E então o que o ASP.NET Core faz automatimente, sem a gente ver?

- Ele cria uma instância de `IProdutoRepository` (através da classe `ProdutoRepository`)
- Ele cria uma instância de `PedidoController`, passando a instância de `IProdutoRepository` criada no passo 1.

Ou seja, esse parâmetro é injetado `automaticamente` sempre que o ASP.NET Core cria uma nova instância do `PedidoController`.

Qual seria a alternativa a isso?

Sem injeção de dependência, teríamos que criar as instâncias manualmente, usando o **operador new**. Teríamos que informar a **classe concreta**, em tempo de desenvolvimento, o tempo todo. Com injeção de dependência, podemos simplesmente definir parâmetros nos construtores, que são **interfaces e não classes concretas**. Por que isso é importante? Porque quando programamos com interfaces fazemos com que as classes **não precisem conhecer** as classes concretas que vão ser criadas, apenas as interfaces. Isso é bom porque **diminui o acoplamento** e a dependência entre classes.

Um exemplo de injeção de dependência: Imagine que você tenha uma aplicação que gera logs sobre a atividade do usuário no seu site. Você então utiliza injeção de dependência, fazendo seu programa chamar métodos da interface `IUserLog`. Inicialmente, você define que os logs vão ser gravados em arquivo texto, através de uma classe `FileLog` (que implementa `IUserLog`). Mas depois de um tempo, você decide gravar os logs em banco de dados, usando uma outra classe chamada `SQLServerLog` (que também implementa `IUserLog`).

Como você faria essa mudança? Com injeção de dependência, você pode simplesmente mudar uma única linha de configuração, indicando que `IUserLog` deve ser instanciada como `SQLServerLog`, e não mais como `FileLog`. Agora, **sem injeção de dependência** você teria que substituir todas as chamadas da classe `FileLog` para `SQLServerLog`.