05

Por que parâmetro como interface e não como classe concreta?

Quando definimos que os parâmetros de métodos e construtores de uma determinada classe são **classes concretas**, corremos o risco de *"engessar"* as relações entre as classes.

É uma boa prática **programar para interfaces**, pois isso diminui o *acoplamento* entre as classes, isto é, *diminui* a dependência entre elas. Por exemplo, imagine esta classe:

```
public class Automovel()
{
    public Automovel(MotorAGasolina motor)
    {
    ...
    }
}
```

O que acontece quando você precisa trocar `MotorAGasolina` por uma outra classe, chamada `MotorAAlcool`, que implementa os mesmos métodos e interfaces? Nesse caso, você precisa mudar a classe `Automovel`:

```
public class Automovel()
{
    public Automovel(MotorAAlcool motor)
    {
    ...
    }
}
```

Mas isso não é bom, pois você está **modificando uma classe que funciona perfeitamente**, apenas para trocar o tipo de um parâmetro!

Isso representa uma violação de um princípio conhecido como **"open-closed principle"**, isto é, **aberto para extensibilidade, fechado para modificação**. De acordo com esse princípio, você deveria projetar sua classe para que as mudanças, como alterações nos tipos de parâmeros, não necessitem de alterações na classe `Automovel`.

Mas quando você estabelece que a interface utilizada é de uma determinada interface, você já definiu um **contrato** entre a classe `Automovel` e o tipo de classe que ela recebe como parâmetro do construtor:

```
public class Automovel()
{
    public Automovel(IMotor motor)
    {
    }
}
```

A partir daí, não importa qual é exatamente a classe recebida como parâmetro do construtor de `Automovel`, desde que ela implemente a interface `IMotor`.