Solid — S.R.P — Single Responsibility Principle

THIAGO ARAGÃO JANUARY 12, 2019

Única é o primeiro e um dos mais importantes princípios da orientação a objetos (OOP).

O princípio tem como objetivo atingir as classes e suas responsabilidades dentro do sistema, é um dos mais fáceis de se entender e aplicar e tem como premissa a seguinte afirmativa:

ou

Realizando uma breve analise, podemos concluir que se uma classe deve ter um único motivo para ser modificada, logo, essa classe só deve ter uma única responsabilidade.

Indo um pouco mais fundo, podemos dizer que para o sistema esteja de acordo com esse princípio, cada responsabilidade deve ser uma classe e cada classe deve ter uma única responsabilidade. Vejamos a classe abaixo:

Aqui temos uma classe fictícia, que apesar de compilar e rodar, tem uma série de responsabilidades estas que não são inerentes a ela:

- Validar dados do Cliente;
- Gerar Imposto;
- Salvar Cupom Fiscal;
- Imprimir Cupom Fiscal;
- Enviar Cupom por e-mail;

Opa, mas se ela tem 5 responsabilidades ela tem 5 motivos para ser modificada. Certo?

Essa quebra de responsabilidades pode gerar uma série de dores de cabeça.

Vamos supor que o método *ValidarDadosDoCliente()* necessitasse ser alterado e após a alteração, um bug não identificado tenha sido publicado em produção.

Teríamos um cenário drástico, pois toda a estrutura da classe foi comprometida, ou seja, o sistema deixaria de emitir notas fiscais por conta de um bug na validação dos dados do cliente.

Além de gerar problemas graves, este tipo de acoplamento traz complexidade ao desenvolver testes além de deixa-los menos eficazes.

Quando se trabalha com o tipo de arquitetura acima, tem-se a falsa impressão de que o sistema está sendo construído de forma prática e simples, pois se tem poucas classes para se dar manutenção, mas, quando elas começam a se expandir, é que vemos como o sistema ficou muito mais complexo e de difícil manutenção.

Quando trabalhamos com SRP tem-se o sentimento de estarmos criando muitas classes para pouco uso, mas quando as funcionalidades do sistema crescem, podemos perceber que a expansão nada mais é do que criar novas classes nos lugares corretos e conecta-las de forma coesa.

Ok, mas o que eu deveria fazer para tornar essa classe coesa e fazê-la seguir a premissa do SRP (Single Responsibility Principle)?

Simples, separar as responsabilidades da classe tornando-as únicas:

Vamos ver os ganhos que tivemos aplicando SRP a classe?

- Facilidade de manutenção e evolução do código;
- Código limpo e de fácil entendimento;
- Facilidade para desenvolvimento de testes;
- Redução do acoplamento;
- Complexidade reduzida;
- Coesão das classes;

Vendo a enorme quantidade de benefícios que tivemos com um simples refactoring, não temos como negar a necessidade de aplicarmos o SRP em nossas aplicações.

Por isso mantenha esse princípio nas “pontas dos dedos” e use-o sem medo.

Não deixe de acompanhar os próximos artigos e aprender um pouco mais sobre outros princípios do SOLID.

Comente sobre o artigo, e se gostar, compartilhe em suas redes sociais com seus amigos e parceiros de trabalho :)

Grande [] e até a próxima!!!