**SOLID**

Hoje estarei abordando um dos princípios SOLID, que são princípios que foram introduzidos por Robert C. Martin, ou como é comumente chamado Uncle Bob. Os princpipios SOLID,  são cinco princípios de engenharia de software para alcançar os benefícios NO DESENVOLVIMENTO. Ficou famoso como SOLID, por que são as iniciais dos princípios, e o pessoal percebeu que formava a palavra SOLID e passou a usá-la. Veja abaixo:

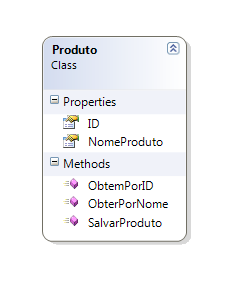
**Principios SOLID**

* **S**igle Responsability Principle
* **O**pen Close Principle
* **L**iskov substitution Principle
* **I**nterface segregation principle
* **D**ependency inversion principle

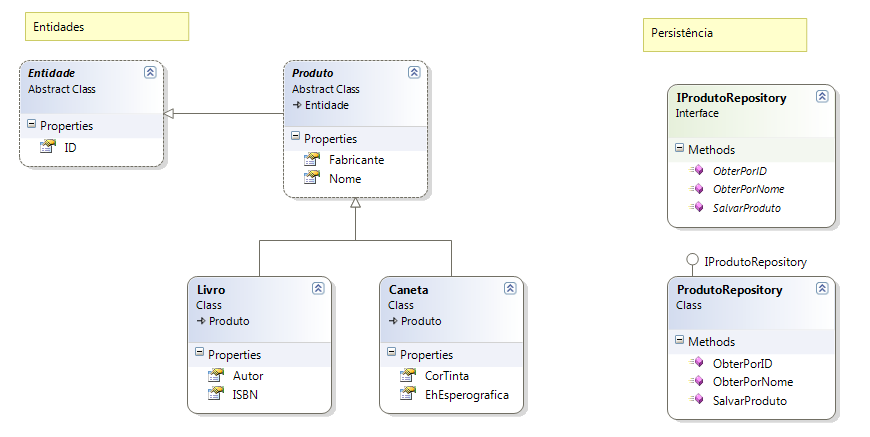
Abordaremos neste artigo o **Sigle Responsability Principle**.

**Combatendo a Classes faz tudo**

Uncle Bob diz eu seu livro clean code, que se uma classe tem mais de um motivo para ser alterada ela já está ferindo o princípio da responsabilidade única. As classes devem ter apenas uma responsabilidade, realizar somente ela e realizá-la bem. Observe o diagrama abaixo:

**Figura 1.** Classe com muitas responsabilidades

O que você vê de arrado na classe acima? Concorda que ela esta assumindo responsabilidades que não são suas? Concorda que ela esta fazendo coisas demais? Ela além de suas propriedades, obtem por id, obtem por nome e ainda salva; diriamos que é uma geniuna classe “canivete suiço”. E geralmente, um modelo de um sistema não tem somente uma classe, mais várias e se  todas estivessem assim? Teriamos vários problemas, para dar manutenção e evoluir. Agore veja o exemplo abaixo, refatorado e contemplando o princípio da responsabilidade única.

**Figura 2.** Classes com responsabilidades bem definidas

Note que temos um cenário completamente diferente, desacolplado e com cada “peça” realizando somente o seu papel. Temos uma classe Entidade, imaginando que estejamos seguindo as premissas do DDD( Domain Drive Design ), uma classe produto que herda de entidade e possui propriedades comuns a todos os produtos, como por exemplo nome e fabricante e classes que herdam de produto e tem suas particularidades em cada uma delas; note que estas classes possuem apenas propriedades e poderiam também, ter métodos e eventos, mais que tratassem apenas do negócio e não de persistência como vimos no exemplo anterior. Na hora de persistir, temos classes apropriadas que vão pegar este objeto de negócio e salvar no banco de dados, ou no repositório destinado a armazená-los. Lembrando que estamos utilizando um padrão de arquitetura chamado repository pattern.

**O Mandamento do princpipio da responsabilidade única**

Uma classe deve fazer apenas uma coisa, deve fazê-la bem e deve fazer somente ela. Se uma classe tem mais de um motivo para ser alterada, ela não segue este princípio. Se ao se referir a uma determinada classe, você diz por exemplo: “minha classe tem as informações do cliente e salva o mesmo no banco de dados” perceba que o **“e”** na frase, indica mais de uma responsabilidade, ferindo assim o SRP( single responsability rpinciple ).

Aplicar o princípio da responsabilidade única, é importante para uma arquitetura madura e sustentável. Quando começamos a dar valor a princípios como este, vemos que estamos amadurecendo e fazendo melhor o que gostamos de fazer: software.