## Uma breve introdução ao S.O.L.I.D.

Os princípios **SOLID** para programação e design orientados a objeto são de autoria de *Robert C. Martin* (*mais conhecido como Uncle Bob*) e datam do início de 2000. A palavra **SOLID** é um acróstico onde cada letra significa a sigla de um princípio, são eles: **SRP, OCP, LSP, ISP e DIP**.

Os princípios **SOLID** devem ser aplicados no desenvolvimento de software de forma que o software produzido tenha as seguintes características:

- **Seja fácil de manter, adaptar e se ajustar às constantes mudanças exigidas pelos clientes;**
- **Seja fácil de entender e testar;**
- **Seja construído de forma a estar preparado para ser facilmente alterado com o menor esforço possível;**
- **Seja possível de ser reaproveitado;**
- **Exista em produção o maior tempo possível;**
- **Que atenda realmente as necessidades dos clientes para o qual foi criado;**


| | | |
|---|---|---|
| **SRP** | The Single Responsibility Principle<br>Principio da Responsabilidade Única | Uma classe deve ter um, e somente um, motivo para mudar.<br>A class should have one, and only one, reason to change. |
| **OCP** | The Open Closed Principle<br>Princípio Aberto-Fechado | Você deve ser capaz de estender um comportamento de uma classe, sem modificá-lo.<br>You should be able to extend a classes behavior, without modifying it. |
| **LSP** | The Liskov Substitution Principle<br>Princípio da Substituição de Liskov | As classes derivadas devem poder substituir suas classes bases.<br>Derived classes must be substitutable for their base classes. |
| **ISP** | The Interface Segregation Principle<br>Princípio da Segregação da Interface | Muitas interfaces específicas são melhores do que uma interface geral<br>Make fine grained interfaces that are client specific. |
| **DIP** | The Dependency Inversion Principle<br>Princípio da inversão da dependência | Dependa de uma abstração e não de uma implementação.<br>Depend on abstractions, not on concretions. |

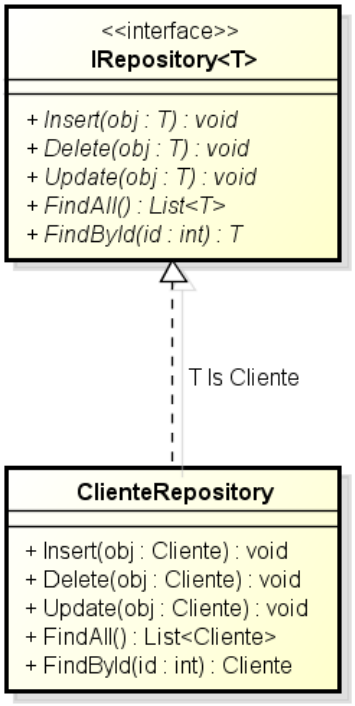

# Princípio da Segregação de Interfaces

"Faça interfaces de fina granularidade,
que sejam específicas para quem vai utilizá-las";
Muitas interfaces específicas são melhores do que uma única interface geral.

O Princípio da Segregação de Interface nos alerta quanto à dependência em relação a "interfaces gordas", forçando que classes concretas programem métodos desnecessários e causando um acoplamento grande entre todos os clientes. Ao usarmos interfaces mais específicas, quebramos esse acoplamento entre as classes clientes, além de deixarmos as implementações mais limpas e coesas.

Esse principio ajuda a evitar a criação de *fat interfaces*(interfaces gordas), termo utilizado para interfaces com mais funcionalidades do que o necessário. Classes que implementam uma interface assim não são coesas. As interfaces podem ser divididas em grupos de métodos, e cada grupo atende uma conjunto diferentes de classes, cada classe pode implementar apenas as funcionalidades que fazem sentido.

**Solução 01: Ferindo o padrão ISP**

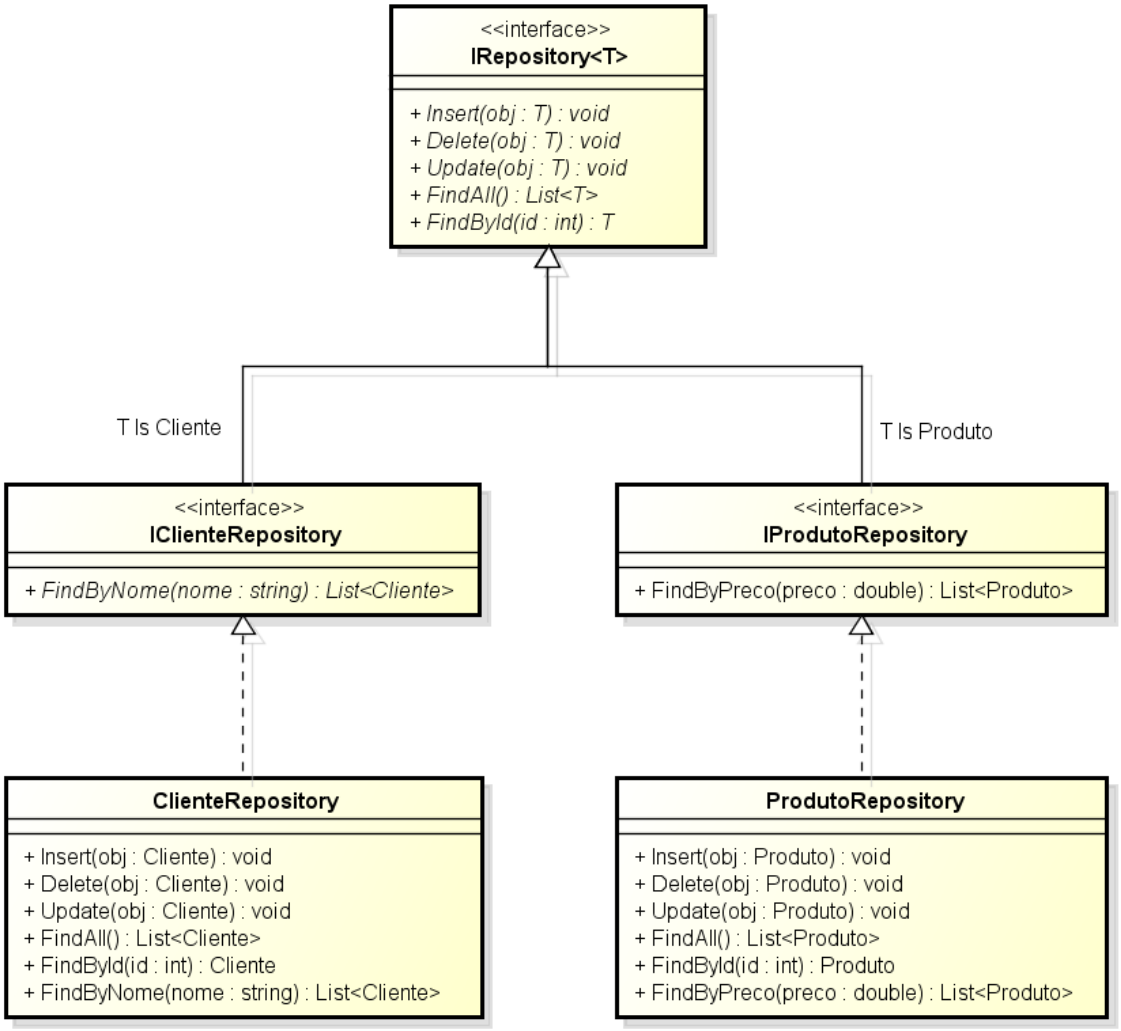


O problema como o modelo acima é justamente o fato da classe ClienteRepository implementar diretamente a interface IRepository. A pergunta que devemos fazer é: "e se cliente precisar de mais operações além daquelas definidas na interface e estas operações são somente voltadas para cliente?".

Como vimos, uma única interface global nunca irá poder atender às necessidades especificas de cada classe que a implementar.

A solução para tal seria granular as operações em diversas interfaces que correspondam a cada entidade especificamente e generalizar as operações comuns em uma interface genérica.

### Solução 02: Aplicando o principio de segregação de interfaces




Uma boa dica para evitar a quebra desse principio é sempre que adicionar um método em uma interface, analise quem implementa essa interface e se aquele método faz sentido para todas as classes que implementam. Se tiver sentido adicione nessa interface sem nenhum problema, caso não faça sentido crie um outra interface(ou verifique se faz sentido em outra interface já existente) para adicionar o método que você precisa implementar.

Indicadores de quebra do principio:

- Métodos de classes, implementados com base em uma interface, retornando valores padrões ou jogando exceções
- Implementações de métodos que não fazem sentido para a classe
- Pouco sentido nas interfaces que existem no sistema
- Muita alterações no código para adicionar um novo método na interface.
- Ao chamar um método em uma classe não ter certeza se ele foi realmente implementado(isso acontece e bastante)

## Codificando:

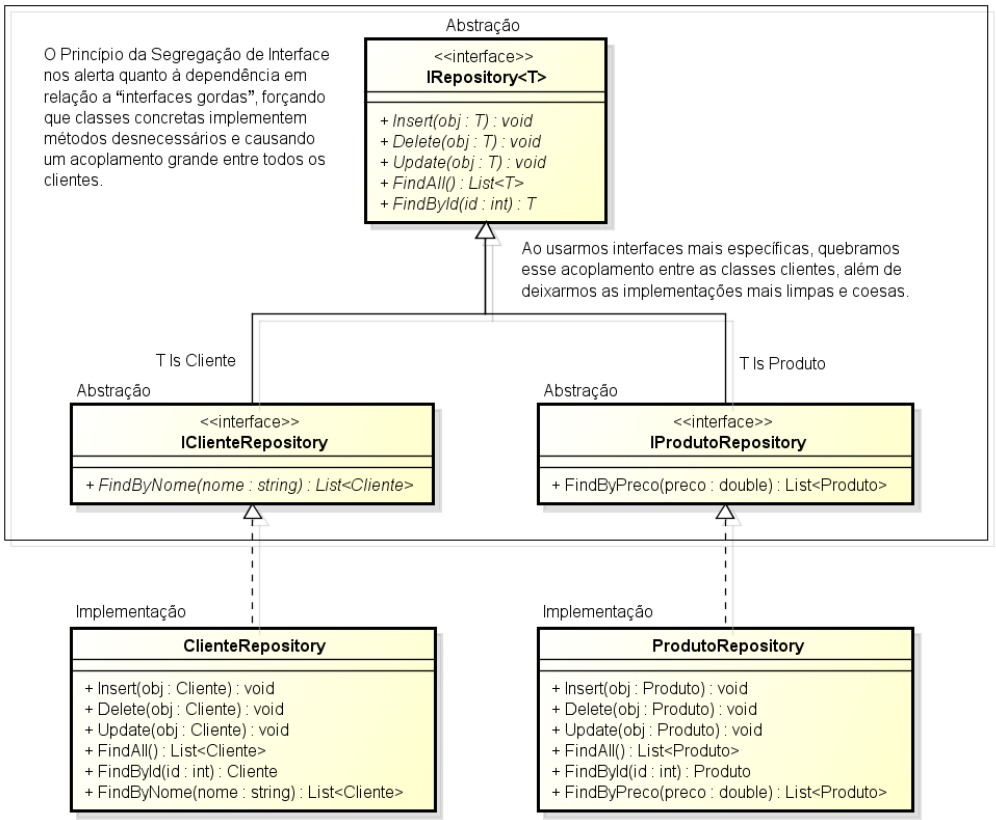
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ISP.Contracts
{
    public interface IRepository<T>
    {
        void Insert(T obj);
        void Update(T obj);
        void Delete(T obj);
        List<T> FindAll();
        T FindById(int id);
    }
}
```

```
using ISP.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ISP.Contracts
{
    public interface IClienteRepository : IRepository<Cliente>
    {
        List<Cliente> FindByNome(string nome);
    }
}
```

```
using ISP.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ISP.Contracts
{
```

```
    public interface IProdutoRepository : IRepository<Produto>
    {
        List<Produto> FindByPreco(double preco);
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ISP.Entities;
using ISP.Contracts;

namespace ISP.Impl
{
    public class ClienteRepository : IClienteRepository
    {
        public void Insert(Cliente obj)
        {
            throw new NotImplementedException();
        }

        public void Update(Cliente obj)
        {
            throw new NotImplementedException();
        }

        public void Delete(Cliente obj)
        {
            throw new NotImplementedException();
        }

        public List<Cliente> FindAll()
        {
            throw new NotImplementedException();
        }

        public Cliente FindById(int id)
        {
            throw new NotImplementedException();
        }

        public List<Cliente> FindByNome(string nome)
        {
            throw new NotImplementedException();
        }
    }
}

using ISP.Contracts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ISP.Entities;

namespace ISP.Impl
{
```

```
        public class ProdutoRepository : IProdutoRepository
        {
            public void Insert(Produto obj)
            {
                throw new NotImplementedException();
            }

            public void Update(Produto obj)
            {
                throw new NotImplementedException();
            }

            public void Delete(Produto obj)
            {
                throw new NotImplementedException();
            }

            public List<Produto> FindAll()
            {
                throw new NotImplementedException();
            }

            public Produto FindById(int id)
            {
                throw new NotImplementedException();
            }

            public List<Produto> FindByPreco(double preco)
            {
                throw new NotImplementedException();
            }
        }
    }
```



**"Classes não devem ser forçados a implementar interfaces que eles não usam."**

Quando estamos criando interfaces temos que tomar certo cuidado e não pensar em generalizar muito as nossas Interfaces, porque uma interface muito genérica com certeza vai nos obrigar a programar métodos ou propriedades que talvez a nossa entidade não utilize, isso além de deixar o código feio, viola o princípio da segregação de interface e dependendo do caso pode até causar uma exception na sua aplicação, então cuidado ao criar as interfaces amplas demais.

**Fontes de Estudo:**

- http://blog.thedigitalgroup.com/rakeshg/2015/05/06/solid-architecture-principle-using-c-with-simple-c-example/
- http://ntakashi.net/principios-solid-principio-da-segregacao-de-interface-isp/
- https://robsoncastilho.com.br/2013/04/14/principios-solid-principio-da-segregacao-de-interface-isp/