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

Princípio da Substituição de Liskov

"Classes derivadas devem poder ser substituídas por suas classes base"

O Princípio de Substituição de Liskov leva esse nome por ter sido criado por Barbara Liskov, em 1988. Sua definição mais objetiva diz que: *"Classes derivadas devem poder ser substituídas por suas classes base".*
Que é uma forma mais simples de explicar a definição formal de Liskov: *"Se para cada objeto o1 do tipo S há um objeto o2 do tipo T de forma que, para todos os programas P definidos em termos de T, o comportamento de P é inalterado quando o1 é substituído por o2 então S é um subtipo de T".*

VIOLAÇÃO MAIS SUTIL (O CLÁSSICO "QUADRADO É UM RETÂNGULO")

Vejamos um exemplo sutil de violação do LSP. Nele, até conseguimos usar a classe-base no lugar da derivada, em termos de compilação, mas, em tempo de execução, teremos um comportamento inesperado (leia-se "bug"). Segue o design, onde um Quadrado é uma classe derivada de Retângulo:

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

namespace LSP.ModoErrado
{
    public class Retangulo
    {
        public virtual double Altura { get; set; }
        public virtual double Comprimento { get; set; }
        public double Area
        {
            get { return Altura * Comprimento; }
        }
    }
}
```

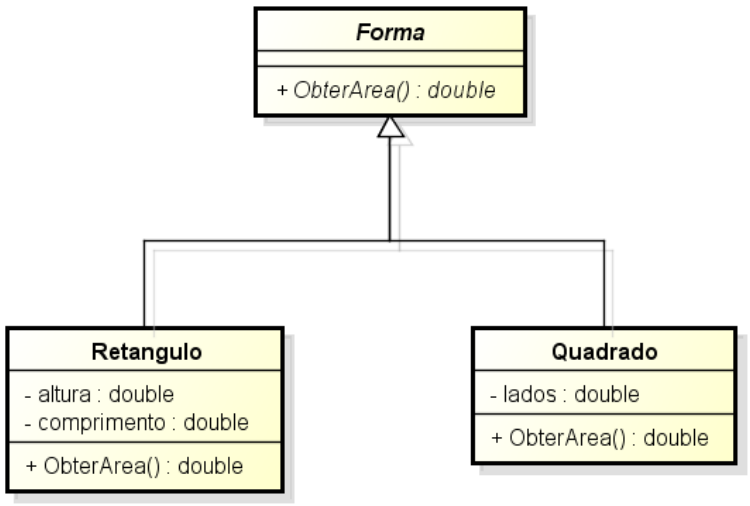
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace LSP.ModoErrado
{
    public class Quadrado : Retangulo
    {
        public override double Altura
        {
            get
            {
                return base.Altura;
            }
            set
            {
                base.Altura = base.Comprimento = value;
            }
        }

        public override double Comprimento
        {
            get
            {
                return base.Comprimento;
            }
            set
            {
                base.Comprimento = base.Altura = value;
            }
        }
    }
}
```

Percebam acima que o Quadrado sobrescreve Altura e Comprimento para manter sua regra de que ambos devem ser iguais. Com isso, a classe derivada viola uma regra estabelecida na classe base: a de que altura e comprimento variam independentemente. Neste caso um quadrado não é um retangulo, e ao aplicarmos o princípio "É Um" da herança de forma convencional vimos que ele não funciona para todos os casos. A instância da classe Quadrado quando usada quebra o código produzindo um resultado errado e isso viola o principio de Liskov onde *uma classe filha (Quadrado) deve poder substituir uma classe base (Retangulo).*

SOLUÇÃO PARA O PROBLEMA DO "QUADRADO É-UM RETANGULO":



O principio da Substituição de Liskov Significa dizer que classes derivadas devem poder substituídas por suas classes base e que classes base podem ser substituídas por qualquer uma das suas subclasses. Uma subclasse deve sobrescrever os métodos da superclasse de forma que a funcionalidade do ponto de vista do cliente continue a mesma.

O princípio da substituição de Liskov nos mostra que devemos tomar cuidado ao fazer uso da herança, devemos verificar se o polimorfismo faz mesmo sentindo, ou seja, se qualquer subclasse pode ser utilizada no lugar da superclasse. Caso não significa dizer que a herança está sendo utilizada de forma inadequada.

Codificando:

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

namespace LSP.ModoCorreto
{
    public abstract class Forma
    {
        public abstract double Area { get; }
    }
}
```

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

namespace LSP.ModoCorreto
{
    public class Retangulo : Forma
    {
        public double Altura { get; set; }
        public double Comprimento { get; set; }

        public override double Area
        {
            get { return Altura * Comprimento; }
        }
    }
}
```

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

namespace LSP.ModoCorreto
{
```

```
    public class Quadrado : Forma
    {
        public double Lados { get; set; }

        public override double Area
        {
            get { return Lados * Lados; }
        }
    }
}
```

O principio **LSP** é uma extensão do **Princípio de Aberto e Fechado (OCP)** e isso significa que temos que ter certeza de que as novas classes derivadas estão estendendo as classes base, sem alterar seu comportamento.

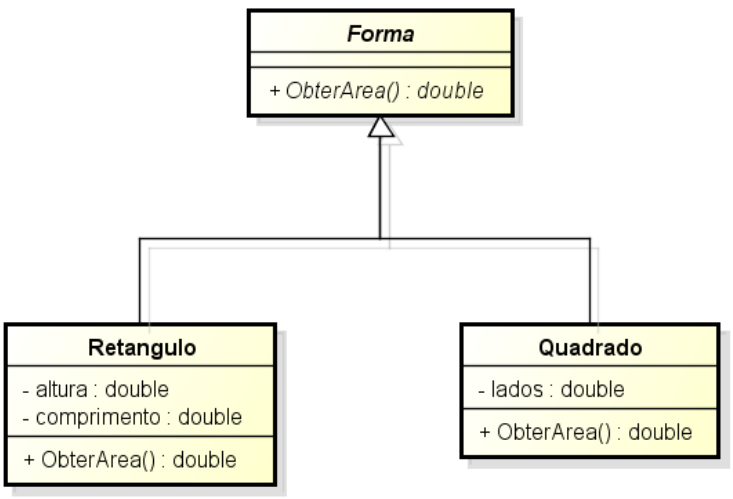
Por que o Princípio da Substituição de Liskov é importante?

1. Porque se não, as hierarquias de classe seriam uma bagunça. Pois podem ocorrer comportamentos estranhos quando uma instância da subclasse for passada como parâmetro para um método.
2. Porque se não, testes de unidade para a superclasse nunca teria sucesso para uma subclasse.

Temos disponíveis várias técnicas para resolver ou evitar o problema de violação do princípio de Liskov, onde podemos usar alguns padrões de projeto no nosso código.

O princípio de substituição de Liskov define que os objetos de uma determinada hierarquia podem ser substituídos por qualquer um dos seus subtipos. Sendo assim a noção de subtipo passa a ser determinada pela noção de substituição.

Se uma hierarquia de classes cujos subtipos não podem ser substituídos um pelos outros, é uma hierarquia errada, segundo Barbara Liskov (esta senhora simpática ai ao lado) e Jeannette Wing, criadoras deste princípio.



Uma vez que Retangulo e Quadrado herdam da mesma classe Forma, tecnicamente é possível substituí-los um pelo outro.

Porem, no exemplo anterior quando Quadrado herda Retangulo este modifica a sua implementação forçando altura e comprimento a obterem o mesmo valor. Por este motivo, se for usado uma instância de Quadrado como saberemos qual o valor correto de altura e comprimento? Assim fica comprovado que esta hierarquia não está seguindo o conceito defendido pelo princípio de Liskov.

Para corrigir este problema a hierarquia deveria ser definida como apresentado neste modelo.

Executando:

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

namespace LSP.ModoCorreto
{
    class Program
    {
        static void Main(string[] args)
        {
            var formas = new List<Forma>();

            formas.Add(new Retangulo() { Altura = 10.0, Comprimento = 20.0 });
            formas.Add(new Retangulo() { Altura = 40.0, Comprimento = 60.0 });
            formas.Add(new Quadrado() { Lados = 10.0 });
            formas.Add(new Quadrado() { Lados = 20.0 });

            foreach(var f in formas)
            {
                Console.WriteLine("Area: " + f.Area);
            }

            Console.ReadKey();
        }
    }
}
```

O exemplo do “quadrado é um retângulo” quebra o Princípio de substituição de Liskov, onde uma classe filha deve substituir plenamente uma classe base. Se um conceito era válido na base, deve ser válido na filha. Nesse caso, o conceito é que os lados variam independentemente, o que não foi respeitado pelo quadrado. Em sistemas, se você não respeita o princípio, a aplicação de polimorfismo pode acabar introduzindo bugs horrorosos.

Fontes de estudo:

- http://www.infragistics.com/community/blogs/dhananjay_kumar/archive/2015/06/30/simplifying-the-liskov-substitution-principle-of-solid-in-c.aspx
- http://blog.thedigitalgroup.com/rakeshg/2015/05/06/solid-architecture-principle-using-c-with-simple-c-example/
- https://robsoncastilho.com.br/2013/03/21/principios-solid-principio-de-substituicao-de-liskov-lsp/