FIT - Universidad Católica del Uruguay

Liskov Substitution Principle (LSP)

Un objeto de una clase que implemente el tipo **IPrinter** puede ser asignado a una variable de tipo **IPrinter**. Hasta el momento habíamos declarado variables del mismo tipo que el objeto que le asignábamos, ahora que sabemos que un objeto puede tener más de un tipo, podemos asignar el objeto a una variable de un tipo siempre y cuando la clase de ese objeto tenga al menos ese tipo; esa clase es un subtipo que contiene todos los tipos que el objeto puede tener.

Enunciado

Bárbara Liskov escribió en 1988 el principio de sustitución que lleva su nombre.

> Si por cada objeto o1 de tipo S hay un objeto o2 de tipo T tal que para todos los programas P definidos en términos de T, el comportamiento de P no cambia cuando o1 se sustituye por o2, entonces S es un subtipo de T.

O, dicho de otra forma, el código que envía mensajes a un objeto de un tipo T debe funcionar igual cuando ese objeto es de un subtipo S del tipo T.

El principio LSP[1] es uno de los principios SOLID. Recuerden que término SOLID es un acrónimo mnemónico de cinco principios destinados a hacer que los diseños de software orientado a objetos sean más comprensibles, flexibles y fáciles de mantener. Los principios SOLID son un subconjunto de muchos principios promovidos por Robert C. Martin, su teoría fue introducida por él en su documento de Design Principles and Design Patterns[2]. En la bibliografía está listado el libro “Agile Principles Patterns and Practices In C#” de Robert C. Martin y Martin Micah, de 2007. También te recomendamos consultar este libro.

Ejemplo

En el programa anterior un objeto en la variable printer de tipo **IPrinter** puede ser tanto una instancia de **ConsolePrinter** como de **FilePrinter** y en ambos casos puedo enviar a ese objeto un mensaje `PrintTicket()` ; **ConsolePrinter** y **FilePrinter** son subtipos de **IPrinter**.

Sin embargo, no es suficiente para cumplir con el principio de sustitución que una clase implemente la interfaz **IPrinter** para poder asignar objetos de esa clase a una variable o argumento declarado del tipo **IPrinter**; además es necesario que ese objeto pueda ser usado en lugar de otro del mismo tipo.

Vean esta clase **TimeMachinePrinter**, que implementa la interfaz **IPrinter**.

```
public class TimeMachinePrinter : IPrinter
    {
        public void PrintTicket(Sale sale)
        {
            if (sale != null)
            {
                sale.DateTime = new DateTime(2001, 01, 01);
                Console.WriteLine(sale.GetTextToPrint());
            }
        }
    }
```

> Ver en repositorio »

Agreguemos un objeto de esa clase en nuestro programa y veamos qué sucede:

```
public class Program
{
    …
    public static void Main(string[] args)
    {
        …
        Sale sale = new Sale();
        …
        IPrinter printer;
        printer = new ConsolePrinter();
        printer.PrintTicket(sale);
        printer = new FilePrinter();
        printer.PrintTicket(sale);
+       printer = new TimeMachinePrinter();
+       printer.PrintTicket(sale);
    }
    …
}
```

> Ver en repositorio »

Vean la salida en consola que resulta de ejecutar este programa -además se crea un archivo Ticket.txt que no es mostrado en este documento. ¿Qué ven de raro?

```
Fecha: 9/10/2018 17:49:51
1 de 'Product 1' a $100
2 de 'Product 2' a $200
```

```
3 de 'Product 3' a $300
Total: $1400
Fecha: 1/1/2001 0:00:00
1 de 'Product 1' a $100
2 de 'Product 2' a $200
3 de 'Product 3' a $300
Total: $1400
```

La fecha de la factura que imprime la clase **TimeMachinePrinter** es diferente de la que imprime **ConsolePrinter...** ¡porque la clase TimeMachinePrinter cambia la fecha de la factura cuando la imprime! Este efecto colateral no está dentro de lo esperado; la operación **PrintTicket** de esta clase es polimórfica pero no respecta el principio de sustitución de Liskov, porque cuando la usamos produce un efecto colateral inesperado.

[1] *Liskov Substitution Principle.*

[2] *Ver https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf .*