TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

ANALISIS Y DISEÑO DE SOFTWARE

# PRINCIPIO DE SUSTITUCIÓN DE LISKOV (LSP)

# PRINCIPIO DE SUSTITUCIÓN DE LISKOV (LSP)

El Principio de Sustitución de Liskov (LSP), formulado por la científica informática Barbara Liskov, establece que, si una clase **S** es un subtipo de otra clase **T**, entonces los objetos del tipo **T** pueden ser reemplazados por objetos del tipo **S** sin alterar el correcto funcionamiento del programa. En otras palabras, una subclase debe poder ser utilizada en lugar de su clase base, sin causar errores ni comportamientos inesperados (García Carmona, 2012).

Este principio se basa en la correcta aplicación de la herencia y el polimorfismo, dos pilares fundamentales de la programación orientada a objetos. Su meta es garantizar que las extensiones de una clase, conserven la coherencia lógica y contractual del comportamiento original.

## Violación del LSP: un ejemplo problemático

Para ilustrar una violación al LSP, tomar como referencia una jerarquía de clases, relacionada con figuras geométricas:

```
class Rectangulo {
    protected int ancho;
    protected int alto;

    public void setAncho(int a) { ancho = a; }
    public void setAlto(int h) { alto = h; }

    public int getArea() {
        return ancho * alto;
    }
}
```

A continuación, se crea una subclase que representa un cuadrado:

```
class Cuadrado extends Rectangulo {
    @Override
    public void setAncho(int a) {
        ancho = a;
        alto = a;
    }

    @Override
    public void setAlto(int h) {
        ancho = h;
        alto = h;
    }
}
```

En este caso, aunque Cuadrado hereda de Rectángulo, altera el comportamiento esperado de la clase base. Si una función espera trabajar con un Rectángulo, pero recibe un Cuadrado, puede experimentar resultados inconsistentes al asumir que puede establecer un ancho y un alto diferentes. Aquí se infringe el LSP, porque el subtipo no respeta las expectativas del tipo base.

## Ejemplo de cumplimiento del LSP

Una forma más adecuada de modelar estas figuras es definir una interfaz común que represente la capacidad de calcular un área, sin imponer una relación de herencia forzada:

```
interface Figura {
    int getArea();
}
```

```
class Rectangulo Implements Figura {
    private int ancho;
    private int alto;

    public Rectangulo(int a, int h) {
        ancho = a;
        alto = h;
    }

    public int getArea() {
        return ancho * alto;
    }
}
```

```
class Cuadrado Implements Figura {
    private int lado;

    public Cuadrado(int l) {
        lado = l;
    }

    public int getArea() {
        return lado * lado;
    }
}
```

En este diseño, cada figura se comporta de manera coherente y predecible. Ambas clases implementan la interfaz Figura y pueden ser utilizadas indistintamente donde se espere una figura, sin romper el contrato de comportamiento. Esto cumple con el LSP.

## Reglas prácticas para aplicar el LSP

- Una subclase no debe eliminar funcionalidades presentes en la clase base.
- Las precondiciones no deben ser más estrictas en la subclase que en la clase padre.
- Las post condiciones no deben ser más débiles que las de la clase base.
- La subclase debe respetar las invariantes del tipo base.