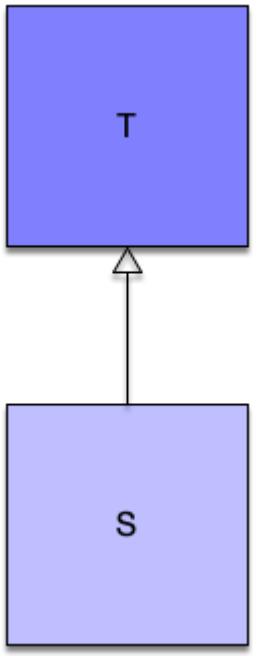
El Principio de Substitución de Liskov es uno de los principios SOLID y hace referencia a cómo usamos la herencia de forma adecuada. El principio dice algo como lo siguiente si S es un subtipo de T , T puede ser reemplazado con objetos de tipo S sin alterar el comportamiento esperado en el programa.



Ufff no es nada sencillo de entender de entrada. Muchas veces me encuentro situaciones en las que la gente me dice bueno lo que quiere decir es que en una relación de herencia tenemos que buscar que la clase hija tenga una relación de “is a” o “es de” . Por ejemplo no puede haber herencia entre Coche y Motor porque un Motor no es un Coche . Sin embargo sí puede haber relación de herencia entre Persona y Deportista ya que un deportista es una persona. Normalmente uno se queda contento con esta respuesta y olvida la definición de la regla de Liskov que es muy compleja. ¿Es esto lo correcto? ¿ Es esto lo que Barbara Liskov decía? . La realidad es que NO , que no hemos entendido nada del enunciado. El Principio de Substitución de Liskov va más allá. Vamos a verlo supongamos que tenemos la clase Persona que dispone de Dni ,Nombre , Apellidos y Tarjeta para realizar pagos.



El código sería algo similar a esto:

```
package com.arquitecturajava;

public class Persona {

	private String dni;
	private String nombre;
	private String apellidos;
	private String tarjeta;

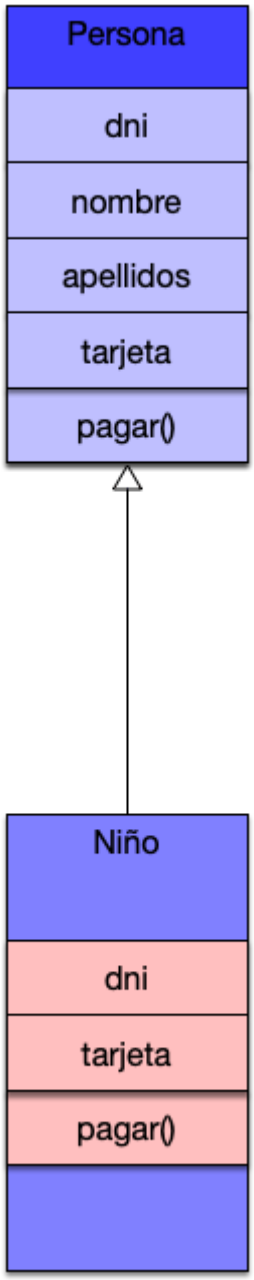
	public String getDni() {
		return dni;
	}
	public void setDni(String dni) {
		this.dni = dni;
	}
	public String getNombre() {
		return nombre;
	}
```

```
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public String getTarjeta() {
		return tarjeta;
	}
	public void setTarjeta(String tarjeta) {
		this.tarjeta = tarjeta;
	}
	public Persona(String dni, String nombre, String apellidos,
String tarjeta) {
		super();
		this.dni = dni;
		this.nombre = nombre;
		this.apellidos = apellidos;
		this.tarjeta = tarjeta;
	}

	public void pagar() {

		System.out.println(&amp;amp;quot;mi dni es
&amp;amp;quot;+ getDni()+ &amp;amp;quot;pago con la
tarjeta&amp;amp;quot;+tarjeta);
	}
}
```

Se trata de una clase sencilla , vamos ahora a usar la herencia y crear una clase hija . En este caso vamos a heredar la clase Niño ya que un niño “es una Persona” por lo tanto es correcto usar la herencia ya que estamos ante una relación de categorización o ¿quizás no?.



El Principio de Substitución de Liskov

Rápidamente comienzan los problemas , el niño no tiene Dni , no tiene tarjeta y gracias a dios no puede pagar nada. Sin embargo sí es un tipo de Persona y cumple con la relación “is a “. ¿Qué es lo que esta sucediendo? . Lo que esta sucediendo es que No estamos aplicando el Principio de Substitución de Liskov . No podemos substituir la clase padre (T) por la clase hija (S) en muchos lugares del programa sin que esto implique un cambio de comportamiento importante. Para que la clase Hija sea simplemente coherente deberíamos hacer algo como:

```
package com.arquitecturajava;

public class Niño  extends Persona{

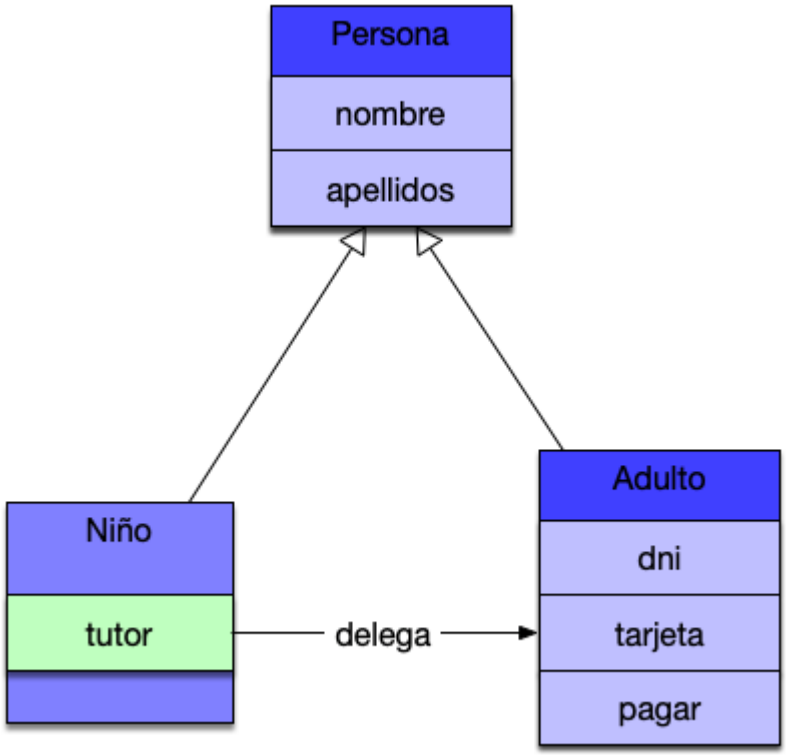
	public Niño(String nombre, String apellidos) {
		super(null, nombre, apellidos, null);
		// TODO Auto-generated constructor stub
	}

	@Override
	public void pagar() {
		// TODO Auto-generated method stub
		throw new RuntimeException(&amp;amp;quot;un niño no
puede pagar&amp;amp;quot;);
	}

}
```

Como se puede observar tenemos que asignar el Dni y la Tarjeta a null y lanzar una

excepción cuando vayamos a pagar . Es evidente que el programa es realmente malo y el uso de la herencia ya nos esta generando problemas importantes. ¿Cómo podemos solventar esto? . Podemos aplicar de forma estricta el Principio de Substitución de Liskov y rediseñar nuestra jerarquía de clases. Vamos a verlo:



En este caso hemos redefinido el concepto de Persona para incluir menos información . Ahora sí que el niño es una Persona ya que siempre tiene nombre y apellidos. Es la clase Adulto la que incorpora el Dni y la tarjeta para pagar. De esta forma todo es más reutilizable. Si queremos que el niño pueda pagar algo lo hará delegando en la clase Adulto que es la que puede hacerlo con la figura de tutor . Veamos el código

```
package com.arquitecturajava.ejemplo2;

public class Adulto extends Persona {
```

```
        public Adulto(String nombre, String apellidos, String dni,
String tarjeta) {
                super(nombre, apellidos);
                this.dni = dni;
                this.tarjeta = tarjeta;
        }

        private String dni;
        private String tarjeta;

        public String getDni() {
                return dni;
        }

        public void setDni(String dni) {
                this.dni = dni;
        }

        public String getTarjeta() {
                return tarjeta;
        }

        public void setTarjeta(String tarjeta) {
                this.tarjeta = tarjeta;
        }

        public void pagar() {

                System.out.println(&amp;amp;quot;mi dni es
&amp;amp;quot; + getDni() + &amp;amp;quot;pago con la
tarjeta&amp;amp;quot; + tarjeta);
```

```
        }
}
```

```
package com.arquitecturajava.ejemplo2;

public class Persona {

        private String nombre;
        private String apellidos;
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public String getApellidos() {
                return apellidos;
        }
        public void setApellidos(String apellidos) {
                this.apellidos = apellidos;
        }
        public Persona( String nombre, String apellidos) {
                super();

                this.nombre = nombre;
                this.apellidos = apellidos;

        }
```

```
}
```

```
package com.arquitecturajava.ejemplo2;

public class Niño  extends Persona{

        private Adulto tutor;

        public Niño(String nombre, String apellidos,Adulto tutor) {
                super( nombre, apellidos);
                this.tutor=tutor;
        }

        public Adulto getTutor() {
                return tutor;
        }

        public void setTutor(Adulto tutor) {
                this.tutor = tutor;
        }

}
```

Este es el código de cada una de las clases nos queda de ver el código del programa main:

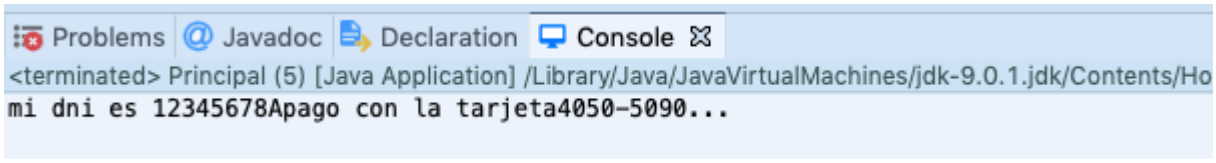
```
package com.arquitecturajava.ejemplo2;

public class Principal {
```

```
    public static void main(String[] args) {
        Adulto adulto = new Adulto("pedro", "perez", "12345678A",
"4050-5090...");
        Niño niño = new Niño("ana", "sanchez", adulto);

        niño.getTutor().pagar();
    }
}
```

Al ejecutar el código el niño podrá realizar una compra si el tutor se la paga . Mucho más razonable.



Este es un ejemplo de cómo usar el Principio de Substitución de Liskov.

Otros artículos relacionados

- JPA @OneToOne y relaciones 1 a 1
- Introducción a Spring Data y JPA
- Java Map for loop , un enfoque moderno
- Java ResultSet un ejemplo Sencillo con Persona