Tabla de Contenidos



Hoy vamos a hablar de Java Optional ifPresent. Cuando trabajamos con el tipo Optional conseguimos mejoras a la hora de trabajar con valores nulos. Veamos un ejemplo sencillo imaginemos que tenemos una clase Libro que contiene una lista de Capítulos.

```
package com.arquitecturajava.ejemplo1;

import java.util.ArrayList;
import java.util.List;

public class Libro {

	private String titulo;
	private List<Capitulo> capitulos=new ArrayList<Capitulo>;
	public Libro(String titulo) {
		super();
		this.titulo = titulo;
	}
	public String getTitulo() {
		return titulo;
	}

	public void setTitulo(String titulo) {
		this.titulo = titulo;
```

```
	}

	public void addCapitulo(Capitulo c) {
		capitulos.add(c);
	}
	public Capitulo buscarCapitulo (String nombre) {
		for (Capitulo c: capitulos) {
			if (c.getNombre().equals("nombre")) {
				return c;
			}
		}
		return null;
	}

}



package com.arquitecturajava.ejemplo1;

public class Capitulo {

	private String nombre;
	private int longitud;
	public Capitulo(String nombre, int longitud) {
		super();
		this.nombre = nombre;
		this.longitud = longitud;
	}
	public String getNombre() {
		return nombre;
	}
```

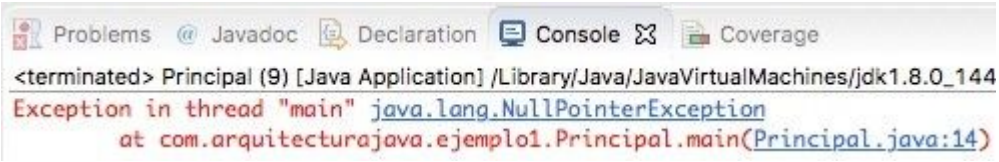
```
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public int getLongitud() {
		return longitud;
	}
	public void setLongitud(int longitud) {
		this.longitud = longitud;
	}
}
```

## Busqueda Capítulo

Disponemos de un método que nos busca un capítulo por nombre. El problema que tiene este método es que en algunas situaciones puede devolver null. El programador que use estas clases puede no darse cuenta. Con lo cual cuando el siguiente programa se ejecute.

```
package com.arquitecturajava.ejemplo1;

public class Principal {

	public static void main (String[] args) {
		Libro l= new Libro("libro A");
		l.addCapitulo(new Capitulo ("capitulo1",50));
		l.addCapitulo(new Capitulo ("capitulo2",20));
		l.addCapitulo(new Capitulo ("capitulo3",40));
		Capitulo busqueda=l.buscarCapitulo("capitulo4");
		System.out.println(busqueda.getNombre());
	}
}
```

Nos encontraremos que el programa fallara y se producirá un NullPointerException.




# Java Optional y Exceptions

Una solución simple para que este error pueda ser controlado es usar manejo de excepciones . De esta forma el código quedaría como sigue:

```
public Capitulo buscarCapitulo (String nombre) throws Exception {
                for (Capitulo c: capitulos) {
                        if (c.getNombre().equals("nombre")) {
                                return c;
                        }
                }
         throw new Exception("objeto no encontrado");
        }
```

Esto implica cambiar el programa principal y añadir un bloque try/catch. La realidad es que es una solución poco elegante y sobre todo muy engorrosa.

```
public class Principal {

        public static void main (String[] args) {
                Libro l= new Libro("libro A");
```

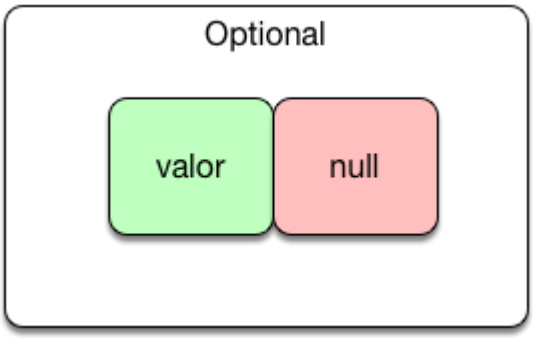
```
                l.addCapitulo(new Capitulo ("capitulo1",50));
                l.addCapitulo(new Capitulo ("capitulo2",20));
                l.addCapitulo(new Capitulo ("capitulo3",40));
                Capitulo busqueda;
                try {
                        busqueda = l.buscarCapitulo("capitulo4");
                        System.out.println(busqueda.getNombre());
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        System.out.println(e);
                }
        }
}
```

## Java Optional ifPresent

Podemos usar el tipo Optional para simplificar esta problemática al máximo.

```
public Optional<Capitulo> buscarCapitulo (String nombre)  {
                for (Capitulo c: capitulos) {
                        if (c.getNombre().equals("nombre")) {
                                return Optional.of(c);
                        }
                }
                return Optional.empty();
        }
```

Recordemos que un Optional es un tipo que permite almacenar dos valores (valor concreto/valor nulo) . Dependiendo de en que situación nos encontramos ejecutaremos una opción u otra.

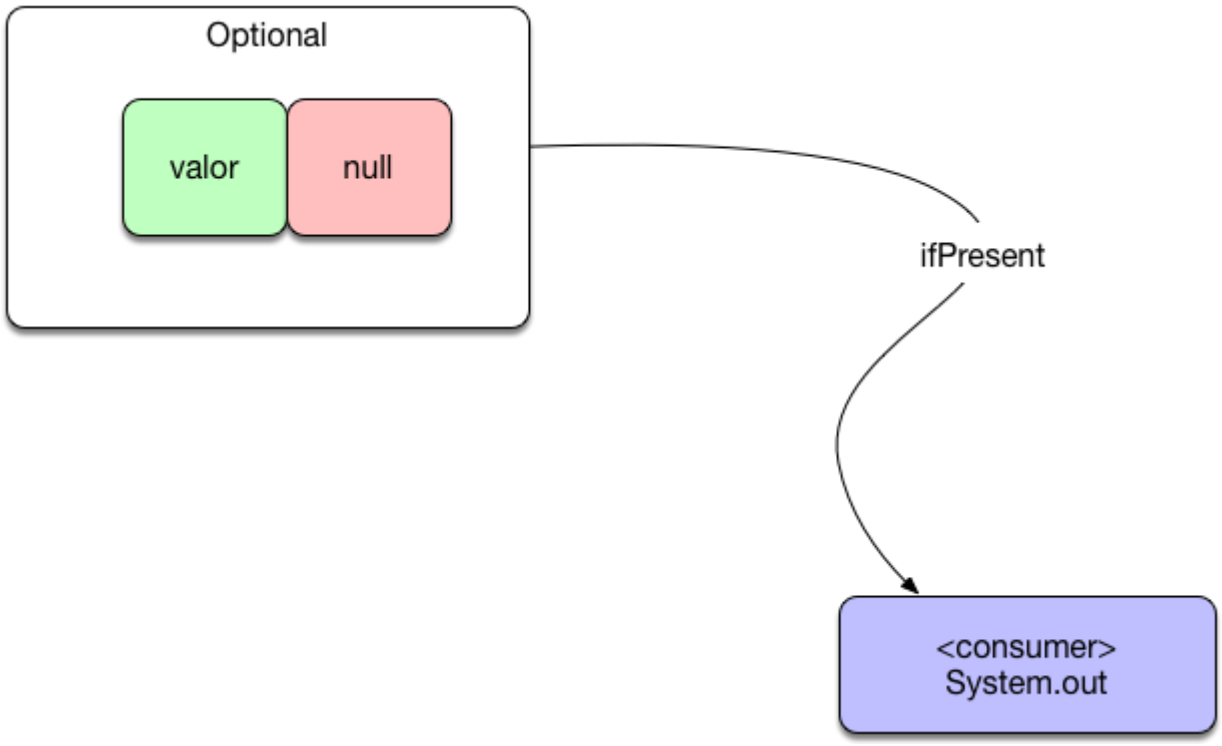



En nuestro caso si el capitulo existe devolvemos un optional con  el capitulo. En caso contrario devolvemos un optional con un valor nulo. De esta forma el programa principal se puede simplificar sobremanera y dejar de hacer uso de una gestión de excepciones abusiva.

```
package com.arquitecturajava.ejemplo3;

import java.util.Optional;

public class Principal {

        public static void main (String[] args) {
                Libro l= new Libro("libro A");
                l.addCapitulo(new Capitulo ("capitulo1",50));
                l.addCapitulo(new Capitulo ("capitulo2",20));
                l.addCapitulo(new Capitulo ("capitulo3",40));
```

```
                    Optional<Capitulo> busqueda;
                        busqueda = l.buscarCapitulo("capitulo2");
busqueda.ifPresent((x)->System.out.println(x.getLongitud()));
        }
}
```

En este caso hemos utilizado un Optional y ademas el método ifPresent que nos permite añadir una expresión lambda como resultado de la operación en el caso de que el valor Optional este presente.




La solución es mucho más elegante y sencilla que la de manejo de excepciones.

Otros artículos relacionados

1. Java Stream Context y simplificación de Streams
2. Java Stream map y estadísticas
3. Java Collections Remove con Java 8
4. Oracle Java Optional