Tabla de Contenidos

- Java NullPointerExcepcions
- Java Optionals al rescate
  - Usando Java Optional
- Otros artículos relacionados

El concepto de Java Optional hace referencia a una variable que puede tener un valor asignado o que puede contener un valor null. En muchas casuísticas nos encontramos con situaciones en las que un valor puede devolver nulo . Ante esta situación los programadores están obligados a comprobar si la variable es null antes de acceder a su valor. Ya que en el caso de ser nula e intentar acceder a algunas de sus propiedades el programa falla y lanza una excepción de java.lang.NullPointerExcepcion Vamos a ver un ejemplo muy sencillo que lo aborde:

```
package com.arquitecturajava;

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

public class Principal {

        public static void main(String[] args) {

                List<Nota> notas= new ArrayList<Nota>();
                notas.add(new Nota("matematicas",3));
                notas.add(new Nota("lengua",10));
                notas.add(new Nota("ingles",5));
                notas.add(new Nota("fisica",7));
                Nota nota= buscarNotaSobresaliente(notas);
```

```
                System.out.println(nota.getValor());
                System.out.println(nota.getAsignatura());
        }

        public static Nota buscarNotaSobresaliente(List<Nota> notas) {

                Nota nota=null;
                for (Nota unaNota:notas) {

                        if (unaNota.getValor()>=9) {
                                nota= unaNota;
                        }
                }
                return nota;
        }

}
```

En este caso tenemos un método que nos busca una Nota en una lista de Objetos . La clase Nota es muy sencilla y almacena el valor y la asignatura:

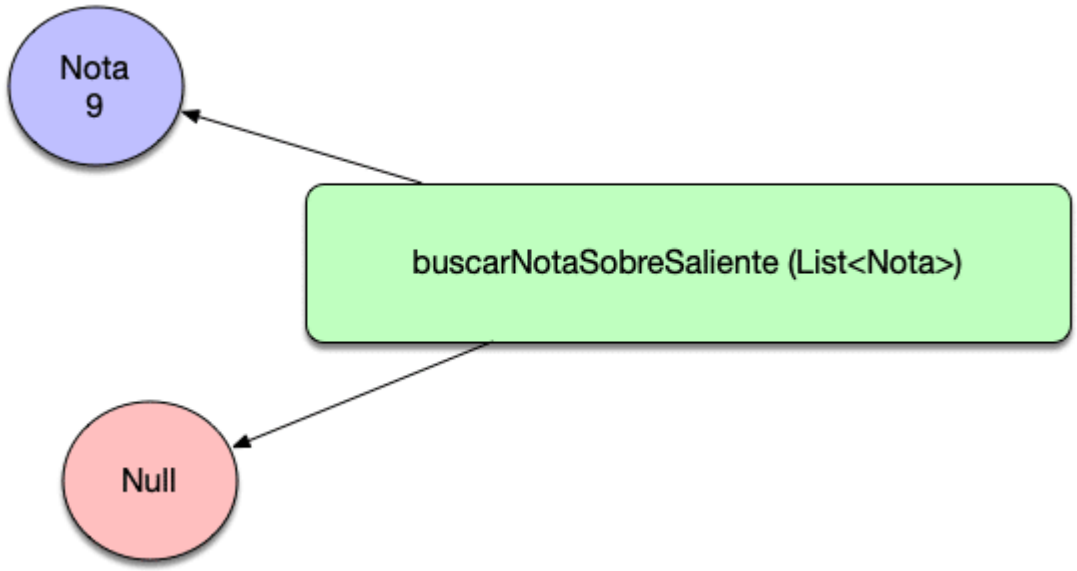
```
package com.arquitecturajava;

public class Nota {

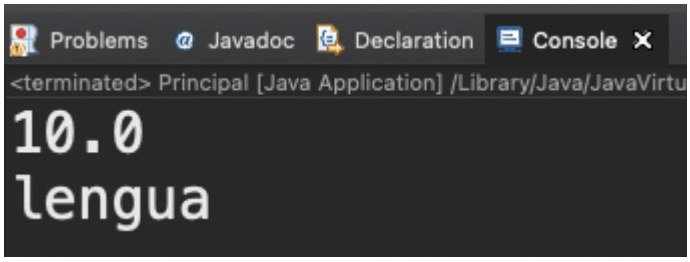
        private String asignatura;
        private double valor;
        public String getAsignatura() {
                return asignatura;
        }
        public void setAsignatura(String asignatura) {
                this.asignatura = asignatura;
```

```
        }
        public double getValor() {
                return valor;
        }
        public void setValor(double valor) {
                this.valor = valor;
        }
        public Nota(String asignatura, double valor) {
                super();
                this.asignatura = asignatura;
                this.valor = valor;
        }
}
```

El programa dispone de un método buscarNotaSobreSaliente que se encarga de buscar la primera Nota sobresaliente de la lista de Notas



Este método puede encontrar o no encontrar la Nota . Por ejemplo en nuestro primer bloque de código si que la encuentra y la imprime por la consola ya que la nota de lengua es un 10.



Ahora bien que pasaría si dejáramos la lista con los siguientes items:

```
notas.add(new Nota("matematicas",3));
notas.add(new Nota("ingles",5));
notas.add(new Nota("fisica",7));
```

En este caso no tenemos una que no contiene ningún sobresaliente y por lo tanto es lógico que devuelva un valor nulo.

```
Exception in thread "main" java.lang.NullPointerException
        at com.arquitecturajava.Principal.main(Principal.java:18)
```

Java NullPointerExcepcions

Tenemos un problema a nivel de código y es que no hemos valorado que el método puede no encontrar ninguna coincidencia y devolver un valor nulo:

```
Nota nota= buscarNotaSobresaliente(notas);
System.out.println(nota.getValor());
System.out.println(nota.getAsignatura());
```

Esto se podría haber solventado de forma sencilla usando una sentencia if que compruebe previamente si existe algún valor nulo.

```
        public static void main(String[] args) {

                List<Nota> notas= new ArrayList<Nota>();
                notas.add(new Nota("matematicas",3));
                notas.add(new Nota("ingles",5));
```

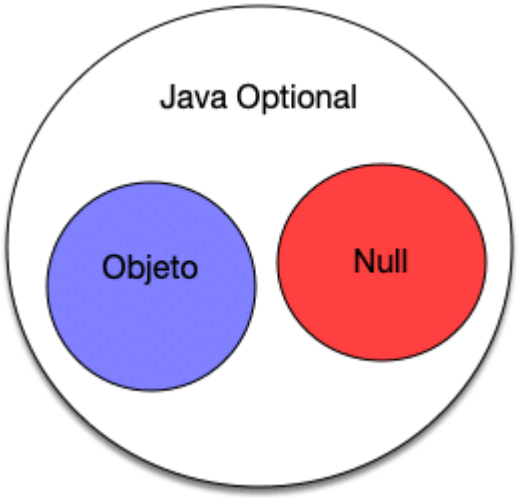
```
                notas.add(new Nota("fisica",7));
                Nota nota= buscarNotaSobresaliente(notas);
                if (nota!=null) {
                        System.out.println(nota.getValor());
                        System.out.println(nota.getAsignatura());
                }
        }
```

De esta forma en el caso que nos ocupa simplemente no imprimirá ningún valor ya que no existe una nota mayor que 9. El problema es que muchas veces no sabemos si siempre se va a devolver una Nota o podemos devolver un valor a nulo . Ese es un problema importante.

Java Optionals al rescate

¿Que es un Java Optional? . Un Java Optional es un tipo de variable que que puede almacenar dos valores .

1. El primer valor es un Objeto que nosotros necesitamos utilizar .
2. El segundo valor es un valor “vacio” o empty en el caso de que nos encontremos con una ausencia de un valor concreto y queramos informar al programador de que estamos ante dicha situación.



Vamos a verlo en acción cambiando el código actual por un código que soporte Optional y simplifique al desarrollador la forma de trabajar con valores nulos:

```
package com.arquitecturajava;

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

public class Principal2 {

	public static void main(String[] args) {

		List<Nota> notas= new ArrayList<Nota>();
		notas.add(new Nota("matematicas",3));
		notas.add(new Nota("ingles",5));
		notas.add(new Nota("fisica",7));
		Optional<Nota> oNota= buscarNotaSobresaliente(notas);
		if (oNota.isPresent()) {
			Nota nota=oNota.get();
			System.out.println(nota.getValor());
			System.out.println(nota.getAsignatura());
		}
	}

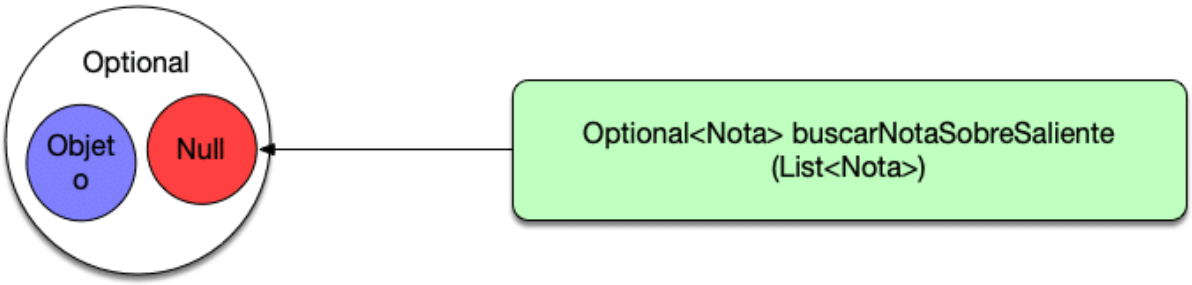
	public static Optional<Nota>
buscarNotaSobresaliente(List<Nota> notas) {

		for (Nota unaNota:notas) {
			if (unaNota.getValor()>=9) {
				return Optional.of(unaNota);
```

```
                }
            }
            return Optional.empty();
        }

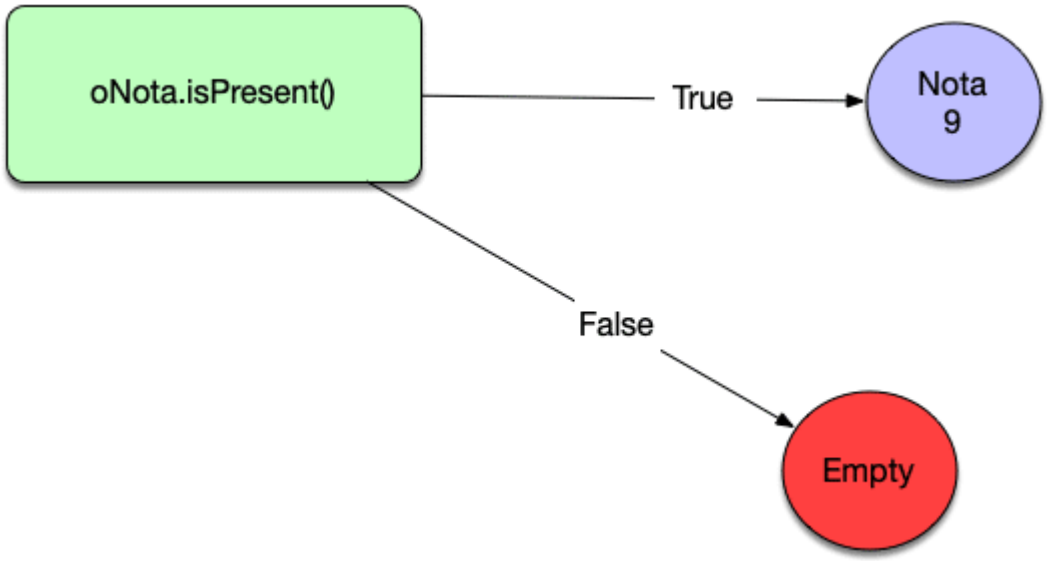
}
```

En este caso hemos cambiado el método de retorno de buscarNotaSobreSaliente por un Optional y este tipo de Objeto encapsula dentro de él dos valores , por un lado el null que en este caso sería un “empty” o lo que es lo mismo no disponemos de un valor. Por el otro lado el objeto que si se encontró en la consulta:



Usando Java Optional

De esa forma con una estructura if , podemos orientar al programador a que accede al valor del optional pero primero pregunte por él utilizando el método isPresent(). Este método nos devuelve true o false dependiendo si el Optional contiene un valor o esta vacío:



Acabamos de ver una introducción sencilla al tipo Optional y como podemos usarlos:

Otros artículos relacionados

- ¿Que es un Java Stream?
- Java Optional Repository y JPA
- Java Optional ifPresent y como utilizarlo
- Java Optional Stream y reference methods
- Java 8 Optional y NullPointerExceptions
- Webinar Java Optionals