OPTIONAL (JAVA SE 11 & JDK 11)

Mauro Serrano Ruiz
Salesianos Triana San Pedro, 41010 Sevilla (Sevilla)
serrano.rumau24@triana.salesianos.eu

1. Introducción a Optional class

La clase **Optional** fue introducida en **Java 8** para simplificar el manejo de *NullPointerException*. Un *NullPointerException* puede aparecer en casos donde se dan un método o propiedad de un objeto con valor nulo ("**null**"), es decir, el concepto de **Java Optional** hace referencia a una variable que puede tener un valor asignado o un valor "**null**".

En muchas ocasiones podemos encontrarnos con la posibilidad de que una **variable** nos devuelva un valor **nulo**, ante esto los desarrolladores 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 *NullPointerException*.

2. ¿Para qué se utiliza la clase Optional?

Una **instancia Optional** es un contenedor que almacena objetos o valores **no nulos**. Se emplea para representar la ausencia de un valor en lugar de utilizar "**null**". Esta clase proporciona varios **métodos útiles** que facilitan el manejo de valores, permitiendo tratarlos como "**disponibles**" o "**no disponibles**". Llevándolo directamente a uno de los métodos que veremos más adelante, si el valor está presente, el método *isPresent()* nos devolverá **true** y en caso contrario nos devolverá **false**.

La clase **Optional** proporciona una solución para verificar el tipo de valor en lugar de comprobar directamente si es "**null**", actúa como un **contenedor** que envuelve el valor. Además de gestionar nulos, ofrece diversos **métodos útiles** para mejorar la claridad y seguridad del código a la hora de compilarlo, como obtener un **valor por defecto** en caso de que el valor sea "**null**" o lanzar una **excepción personalizada**.

En definitiva, **Optional** permite escribir código de forma más **segura**, reduciendo la necesidad de realizar **comprobaciones explícitas de "null"**.

3. Creación de instancias de Optional

En la clase **Optional** existen varias formas de **instanciar objetos**, dependiendo de si queremos manejar **valores nulos o no**.

La primera forma es mediante el método *Optional.of(T value)*, que permite instanciar un objeto **Optional** con un **valor no nulo**. Es importante saber que si el valor proporcionado es "**null**", este método lanzará una excepción *NullPointerException*, por lo que debe utilizarse únicamente cuando se tiene la certeza de que el valor **no será nulo**.

```
Optional<String> optionalConValor = Optional.of("Hola, mundo"); // No permite valores nulos
```

Ante la posibilidad de un **valor nulo**, podemos utilizar *Optional.ofNullable(T value)*. Este método instancia un **Optional** con el **valor que introduzcamos**, y en caso de que este sea **nulo**, devuelve un **Optional vacío**. Esto permite gestionar la ausencia de valor.

Optional<String> optionalConNulo = Optional.ofNullable(null); // Permite valores nulos

Por último, existe la opción de instanciar un **Optional vacío** utilizando *Optional.empty( )*. Este método devuelve una instancia de **Optional** sin ningún valor (**IMPORTANTE:** no confundir **ausencia de valor** con un "**null**"), lo que permite representar la ausencia de datos sin recurrir a "**null**".

Optional<String> optionalVacio = Optional.empty(); // Representa ausencia de valor

4. Métodos

La clase **Optional** proporciona una serie de **métodos** diseñados para facilitar la manipulación de **valores opcionales**, evitando la comprobación manual de posibles valores "**null**". A continuación veremos algunos de los métodos más importantes así como su utilización.

isPresent( ) & isEmpty( )

Uno de los métodos **más utilizados** es *isPresent( )*, que nos permite **verificar** si el Optional contiene o no un **valor**. Este método nos devuelve "**true**" si hay un valor presente y "**false**" en el caso contrario. En **Java 11** se introdujo un **nuevo método**, *isEmpty( )*, cuya función es la misma pero de manera **inversa**, devolviendo un "**true**" si el Optional está **vacío** y viceversa. Estos métodos son útiles a la hora de realizar **comprobaciones** antes de acceder al valor contenido en un **Optional**.

Optional<String> opt = Optional.of("Java");

System.out.println(opt.isPresent()); // Devuelve true

System.out.println(opt.isEmpty()); // Devuelve false

get( )

Este método nos permite **obtener** o acceder al **valor** de un **Optional**. Debe usarse con precaución pues, si el **Optional** está **vacío**, nos devolverá una excepción del tipo **NoSuchElementException**. En estos casos se recomiendo la utilización de los siguientes métodos a explicar, *orElse( )* u *orElseGet( )*.

Optional<String> opt = Optional.of("Ejemplo");

System.out.println(opt.get()); // Imprime "Ejemplo"

orElse(T other) & orElseGet(Supplier<? extends T> supplier)

Ambos métodos permiten **obtener** un **valor alternativo** en caso de que un **Optional esté vacío**. La diferencia entre ellos es que *orElse( )* siempre **evalúa** el valor alternativo, incluso si el Optional contiene valor, mientras que *orElseGet( )* solo lo **genera** si el Optional está **vacío**. Esto es importante en cuanto a términos de **rendimiento**, pués *orElse( )* puede ser útil ante una **expresión ligera**, pero ante una **pesada** lo mejor sería utilizar el método *orElseGet( )*.

```
Optional<String> opt = Optional.empty();

System.out.println(opt.orElse("Valor por defecto")); //  Imprime "Valor por defecto"

System.out.println(opt.orElseGet(() -> "Generado dinámicamente")); //  Imprime "Generado dinámicamente" (Utiliza lambda)
```

orElseThrow( )

Con este método podemos lanzar una **excepción** ante la **ausencia de valor**, en lugar de proporcionar un valor por defecto. *OrElseThrow( )* nos permite especificar una excepción **personalizada** que se lanzará en caso de que el **Optional esté vacío**.

```
Optional<String> opt = Optional.empty();

System.out.println(opt.orElseThrow(() -> new RuntimeException("Valor no presente")));
```

ifPresent(Consumer<? super T> action) & ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)

El primer método nos permite "**configurar**" una **acción** para que sea ejecutada si el **Optional contiene un valor**, una función parecida a la que realiza la estructura *if*. En **Java 9** se agregó el método *ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)*, que permite realizar la **misma tarea** pero en caso de que el **Optional esté vacío** e implementando **lambda**.

```
Optional<String> opt = Optional.of("Hola");

opt.ifPresent(System.out::println); // Imprime "Hola"
```

map(Function<? super T, ? extends U> mapper) y flatMap(Function<? super T, Optional<U>> mapper)

Estos métodos permiten **transformar** el **valor** contenido en un **Optional**. El primero, *map( )*, aplica una **función** al valor, si está presente, y **devuelve** un nuevo **Optional** con el resultado. Por otro lado, *flatMap( )* la transformación ya devuelve un **Optional**, en ese caso **aplana el resultado** y devuelve un solo **Optional**.

```
Optional<String> opt = Optional.of("java");

Optional<String> mayusculas = opt.map(String::toUpperCase); // Transforma "java" en "JAVA"

System.out.println(mayusculas.get()); // Imprime "JAVA"
```

5. Conclusión

La **clase Optional** es muy útil para **evitar errores** cuando trabajamos con posibles valores "**null**". En lugar de hacer comprobaciones manuales, **Optional** nos permite **manejar** de forma más segura la **presencia** o **ausencia de un valor**.

Gracias a sus **métodos**, podemos saber si un **valor está presente** (*isPresent( )*), definir un **valor por defecto** (*orElse( )*), **ejecutar acciones** si el valor existe (*ifPresent( )*) y **transformar** su contenido (*map( )* y *flatMap( )*). Todo esto ayuda a **optimizar el código** y que sea menos propenso a fallos.

6. Fuentes

https://www.tutorialspoint.com/java/java_optional_class.htm

https://www.arquitecturajava.com/que-es-un-java-optional/

https://www.youtube.com/watch?v=vX-JAs73O9o

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Optional.html