**Java 8 Optional-2022**

**Optional isPresent and ifPresent**

**Optional.isPresent()**:  returns true if the given Optional object is non-empty. Otherwise it returns false.

**Optional.ifPresent():** performs given **action** if the given Optional object is non-empty. Otherwise it returns false.

**Usage of isPresent() as true or false** 🡸 Validation

public static Optional getSomething(String input) {  
 return (input != null) ? Optional.*of*(input) : Optional.*empty*();  
}

Optional<String> val = *getSomething*("some value");  
System.*out*.println("Is Present: "+val.isPresent()); 🡸 Prints true

**Usage of ifPresent as an Action (Consumer)** 🡸 An Action

public class Test1 {  
  
 public static void doSomething(String value) {  
 System.*out*.println("Value : "+value);  
 }  
  
 public static void main(String[] args) {  
 String test = "Hello World";  
 Optional<String> strOpt = Optional.*ofNullable*(test);  
 **strOpt.ifPresent**( value -> Test1.*doSomething*(value)); 🡸 **An Action**  
 }  
}

**\*\* Always use Optional.ofNullable when using .ifPresent because in case of Optional.of() will throw NullPointerException if the value is null.**

if String test = null;, then no action will be executed.

Optional<String> opt = Optional.*of*(**"Male"**);  
*//Performing an action*opt.ifPresent( (value) -> System.***out***.println(**"We got :"**+value));

**String val = null;  
Optional.*ofNullable*(val).ifPresent((v) -> System.*out*.println("Value: "+v));**

**Optional.of vs Optional. ofNullable**

I**n case of null** , Optional.of 🡺 throws NullPointerException,

I**n case of null** , Optional.ofNullable 🡺 returns **Optional.empty**

**If the element is not null, then both Optional.of and Optional.ofNullable behavior is same**.

String str = **null**;  
System.***out***.println(Optional.*of*(str)); *🡸* ***NullPointerException***System.***out***.println(Optional.*ofNullable*(str)); *🡸* ***Optional.empty***

String val = **null**;  
System.***out***.println(Optional.*ofNullable*(val).get()); *🡸* ***NoSuchElementException***System.***out***.println(Optional.*of*(val).get()); *// 🡸* ***NullPointerException***

String str = **"Hello World"**;  
System.***out***.println(Optional.*of*(str).get()); *🡸 Hello World*System.***out***.println(Optional.*ofNullable*(str).get()); *🡸 Hello World*

String str = **null**;  
System.***out***.println(Optional.*of*(str).orElse(**"some value"**)); 🡸 **NullPointerException**  
System.***out***.println(Optional.*ofNullable*(str).orElse(**"some value"**)); 🡸 **some value**

**orElse orElseGet() methods**

String nullName = **null**;  
String name = Optional.*ofNullable*(nullName).orElse(**"john"**);  
System.***out***.println(name); 🡸 Prints john

* **orElse()** will always call the given function whether you want it or not, regardless of Optional.isPresent() value
* **orElseGet()** will only call the given function when the Optional.isPresent() == false

**orElse**

**public static** String getOtherValue() {  
 System.***out***.println(**"Get Other value method called"**);  
 **return "Other value"**;  
}

**public static void** main(String[] args) **throws** Exception {  
 String val = **"abcd"**;  
 String name = Optional.*ofNullable*(val).orElse(*getOtherValue*());  
 System.***out***.println(name); 🡸 abcd

}

**OUTPUT**

**Get Other value method called**

abcd

**.orElse() by default calls the getOtherValue() even if it is non null**;

**orElseGet() 🡸 An Action**

**public static** String getHeavyObject() {  
 System.***out***.println(**"Method called "**);  
 **return "Heavy"**;  
}  
  
**public static void** main(String[] args) **throws** Exception {  
 String nullName = **null**;  
 String name = Optional.*ofNullable*(nullName).orElseGet(() -> Test.*getHeavyObject*());  
 System.***out***.println(name);  
}

**OUTPUT**

Method called

Heavy

**public static** String getOtherValue() {  
 System.***out***.println(**"Get Other value method called"**);  
 **return "Other value"**;  
}

String val = **"abcd"**;  
String name = Optional.*ofNullable*(val).orElseGet(() -> *getOtherValue*());  
System.***out***.println(name);

**OUTPUT**

abcd

**orElseThrow()**

**String** name = Optional.ofNullable(nullName).orElseThrow( IllegalArgumentException::**new**);

**filter()**

**Integer** year = 2016;

Optional<Integer> yearOptional = Optional.of(year);

**boolean** is2017 = yearOptional.filter(y -> y == 2017).isPresent();

**Optional.flatMap and Optional.map**

Use**map**if the function returns the object you need, **Map** - wraps the result in an Optional.

Use **flatMap** if the function returns an **Optional**, **~~flatMap~~**~~- returns the 'raw' object~~

For example:

public static void main(String[] args) {

Optional<String> s = Optional.of("input");

System.out.println(s.map(Test::getOutput));

System.out.println(s.flatMap(Test::getOutputOpt));

}

static String getOutput(String input) {

return input == null ? null : "output for " + input;

}

static Optional<String> getOutputOpt(String input) {

return input == null ? Optional.empty() : Optional.of("output for " + input);

}

**Another Type**

**public static** String getOutput(String input) {  
 **return** input == **null** ? **null** : **"output for "** + input;  
}

**public static** Optional<String> getOutputOpt(String input) {  
 **return** input == **null** ? Optional.*empty*() : Optional.*of*(**"output for "** + input);  
}

String input = **null**;  
System.***out***.println(Optional.*ofNullable*(input).map((in) -> *getOutput*(in)));  
System.***out***.println(Optional.*ofNullable*(input).flatMap((in) -> *getOutputOpt*(in)));

**OUTPUT**

Optional.empty

Optional.empty

Both print statements print the same thing.

System.***out***.println(Optional.*of*(10).map(x -> x \* x));  
System.***out***.println(Optional.*of*(10).flatMap(x -> Optional.*of*(x \* x)));  
System.***out***.println(Optional.*of*(10).map(x -> Optional.*of*(x \* x).get()));  
  
System.***out***.println(Optional.<Integer>*empty*().map(x -> x \* x));  
System.***out***.println(Optional.<Integer>*empty*().flatMap(x -> Optional.*of*(x \* x)));  
System.***out***.println(Optional.<Integer>*empty*().map(x -> Optional.*of*(x \* x).get()));

**OUTPUT**

Optional[100]

Optional[100]

Optional[100]

Optional.empty

Optional.empty

Optional.empty