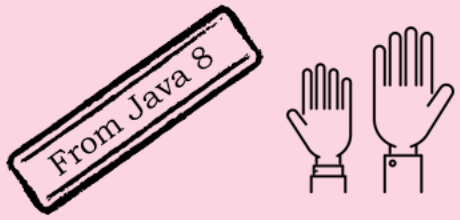
Optional to deal with nulls

eazy bytes



If I ask a question to all the Java developers in the world to raise their hand if they have ever seen NullPointerException in their code, there might be no one who will not raise their hand ☺

- ✓ We as developers spend good amount of time fixing or caring for the NullPointerException in our code and it is a very painful or tedious process always. Below is one of the sample code where we can see an null pointer exception,

```
public String getUserOrderDetails(User user) {
    return user.getOrder().getitem().getName();
}
```

In the above code we may get null pointer exception at any instance like while accessing the order from user/item from order/name from item in case if they are null values. If we have to handle them we will end up writing code like below,

```
public String getUserOrderDetails(User user) {
    if (user != null) {
        Order order = user.getOrder();
        if (order != null) {
            Item item = order.getItem();
            if (item != null) {
                return item.getName();
            }
        }
    }
    return "Not Available";
}
```

Optional to deal with nulls

eazy bytes

- ✓ Addressing the challenge of NullPointerExceptions in Java doesn't involve replacing the language with a new one. Instead, Java offers a library construct to handle this issue gracefully. The java.util package provides the Optional<T> class, which serves as a solution for dealing with NullPointerExceptions. In practice, methods that might not return a value are advised to return an Optional instead of null. This approach enhances code robustness and clarity, providing a more structured way to manage scenarios where a result may be absent.

- ✓ An important, practical semantic difference in using Optionals instead of nulls is that in the first case, declaring a variable of type Optional<String> instead of String clearly signals that a missing value is permitted there. Otherwise you always depends on the business domain knowledge of the user.

- ✓ When a value is present, the Optional class wraps it, if not the absence of a value is modeled with an empty optional returned by the method Optional.empty(). An Optional in Java serves as a container object that can either contain or not contain a non-null value. Its isPresent() method indicates whether it holds a non-null value, returning true if present and false otherwise. The get() method retrieves the non-null value if present but throws a NoSuchElementException if the container is empty.

- ✓
```
Optional<String> optString = Optional.empty(); // Creating an empty optional
Optional<String> optString = Optional.of(strObj); // Optional from a non-null value
Optional<String> optString = Optional.ofNullable(strObj); // If product is null, the resulting Optional object woul d be empty
```

Optional to deal with nulls

eazy bytes

- ✓ Consequently, when a method returns an Optional, it is advisable to check for the presence of a non-null value before calling the get() method. This practice helps prevent a NoSuchElementException from being thrown, offering a more controlled approach than the potential NullPointerException. While the Optional construct doesn't entirely eliminate the possibility of NullPointerExceptions, using it encourages developers to follow a more disciplined and safer approach when dealing with potentially absent values.

- ✓ Advantages of Optional

    - Null checks are not required.
    - No more NullPointerException at run-time.
    - We can develop clean and neat APIs.
    - No more Boiler plate code

Optional to deal with nulls

eazy bytes

- ✓ Important methods provided by Optional in Java 8,
  - `of()` – Returns an Optional describing the given non-null value.
  - `ofNullable()` – Returns an Optional describing the given value, if non-null, otherwise returns an empty Optional.
  - `empty()` - Returns an empty Optional instance
  - `isPresent()` - Returns true if a value is present; otherwise, returns false
  - `get()` - Returns the value wrapped by this Optional if present; otherwise, throws a NoSuchElementException
  - `ifPresent()` - If a value is present, invokes the specified consumer with the value; otherwise, does nothing
  - `orElse()` - Returns the value if present; otherwise, returns the given default value
  - `orElseGet()` - Returns the value if present; otherwise, returns the one provided by the given Supplier
  - `orElseThrow()` - Returns the value if present; otherwise, throws the exception created by the given Supplier
  - `map()` - If a value is present, applies the provided mapping function to it
  - `filter()` - If the value is present and matches the given predicate, returns this Optional; otherwise, returns the empty one