Avoiding Nulls with the Optional Type

Esteban Herrera
JAVA ARCHITECT

@eh3rrera www.eherrera.net

“I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

Tony Hoare

```java
public LoyaltyLevel getCustomerLoyaltyLevel(Customer customer) {
    if (customer != null) {
        LoyaltyCard lc = customer.getLoyaltyCard();
        if (lc != null) {
            return lc.getLevel();
        }
    } else {
        return null;
    }
    return null;
}
```

```
Customer customer = repository.findCustomerById(id);

String name = customer.getName();
```

```
Customer customer = repository.findCustomerById(id);

// ...

String name = customer.getName();
```

Throwing an exception
is not a good solution.

Sometimes, the absence
of data is not really an error.

```
double discount = repository.findCurrentDiscountPercentage();

if (discount > 0) {
    // ...
}
```

What should we return in this case?

```
Customer customer = repository.findCustomerById(id);
```

List

1, 2, 3, 4, 5

The Optional Type

By Using a Type Like Maybe

- You explicitly indicate that something may be null
- There cannot be NullPointerExceptions

Optional

value | null

Creating an Optional

```
// From a non-null object
Optional<Customer> optionalCustomer = Optional.of(customer);

// From an object that may hold a null value
Optional<Customer> optionalCustomer = Optional.ofNullable(customer);

// Creating an empty optional
Optional<Customer> optionalCustomer = Optional.empty();
```

Unpacking a Value from an Optional

```
// It can throw a NoSuchElementException
Customer customer = optionalCustomer.get();

// You can check if it has a value first, but...
if ( optionalCustomer.isPresent() ) { // Java 10 added isEmpty()
  customer = optionalCustomer.get();
} else {
   // ...
}
```

Unpacking a Value from an Optional

```
// To provide a default value
Customer customer = optionalCustomer.orElse( new Customer() );

// To provide a default value via a Supplier
Customer customer = optionalCustomer.orElseGet( () -> new Customer() );

// To provide a default value of type Optional via a Supplier (Java 9+)
Customer customer = optionalCustomer.or( () -> Optional.empty() );

// Similar to get(), you choose the exception to be thrown
Customer cust = optionalCustomer.orElseThrow( RuntimeException::new );
```

Executing Actions with an Optional

```
// Executes the consumer given as argument if a value is present
optionalCustomer.ifPresent( System.out::println );

// Also takes a Runnable to be executed if Optional is empty (Java 9+)
optionalCustomer.ifPresentOrElse(
                  System.out::println,
                  () -> System.out.println("Empty");
);
```

Methods You Should Use

Filter

```
Optional<T> filter(Predicate<? super T> predicate)
```

Filter

```
Customer customer = repository.findCustomer(id);
if (customer != null && customer.getRewardsPoints() > 0) {
    System.out.println("Customer has rewards points");
}
```

Filter

```
repository.findCustomer(id)
    .filter(customer -> customer.getRewardsPoints() > 0)
    .ifPresent(
        customer -> System.out.println("Customer has rewards points")
    );
```

Map

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

Map

```
String name = "";
Customer customer = repository.findCustomer(id);
if (customer != null){
    name = customer.getName();
}
```

Map

```
String name = repository.findCustomer(id)
                .map(customer -> customer.getName())
                .orElse("");
```

FlatMap

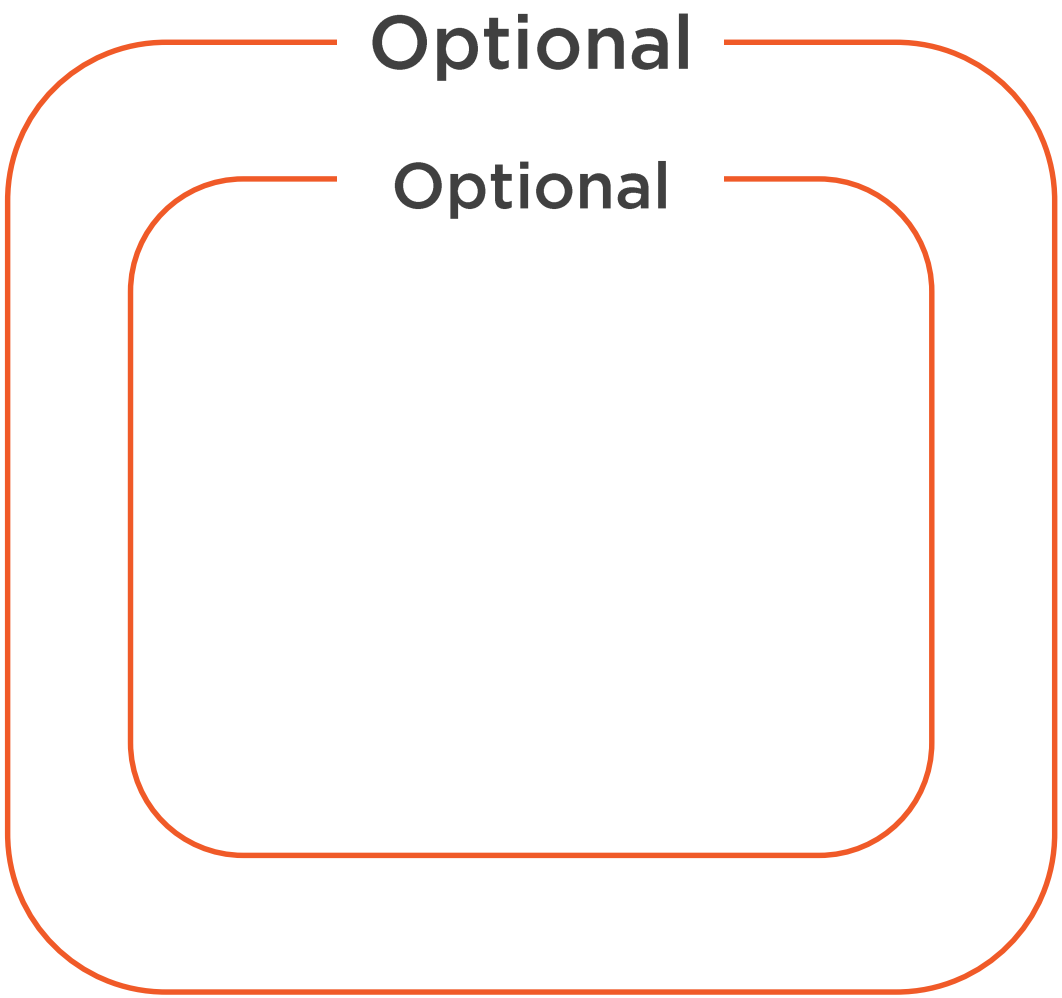
```
String name = repository.findCustomer(id)
                .flatMap(customer -> customer.getName())
                .orElse("");
```

If getName() returns
an Optional, use flatMap

FlatMap

```
Optional<U> flatMap(Function<? super T, Optional<U> mapper)
```

```
String name = repository.findCustomer(id)
                .map(customer -> customer.getName())
                .orElse("");
```


Optional
Optional

If the function returns a plain object, use map.

If the function returns an Optional, use flatMap.

How to Use Java's Optional Type

Using Optional through Composition

- Always start from an Optional
- Apply a chain of filter, map, or flatMap methods
- Use orElse or orElseGet to unwrap the value

Lessons

- Return an Optional to signal that there might be no return value in some cases
- Use map or flatMap to apply a chain of transformation with multiple stages
- Don’t use Optional as a method argument
- Don't be afraid to make your getter methods work with an Optional type instead of null

Implementing a Lift Method

Methods Added to Optional

Java 9	Java 10	Java 11
or ifPresentOrElse stream	orElseThrow	isEmpty

Lifting

Allows you to transform a function of plain types to a function of the same types wrapped in a container type.

Things to Remember

Nulls make the code dishonest

- Just by looking at the signature of a method, we cannot tell whether it returns a null reference or not

Optional explicitly indicates the potential absence of a value

- It is based on the Maybe type you can find in functional languages

Things to Remember

Optional's main methods are filter, map, flatMap, and orElse/orElseGet

- It can be useful to think of an optional as a stream with one element

The best way to use Optional is through composition

- Always start from an Optional
- Apply a chain of filter, map, or flatMap methods
- At the end, use orElse or orElseGet to unwrap the value

Don't use Optional as a method argument

In the Next Module

Handling errors in a functional style