Using Optional Instead of Null

Esteban Herrera
JAVA ARCHITECT

@eh3rrera   www.eherrera.net

```
Book book = service.findBookById(id);

String title = book.getTitle();
```

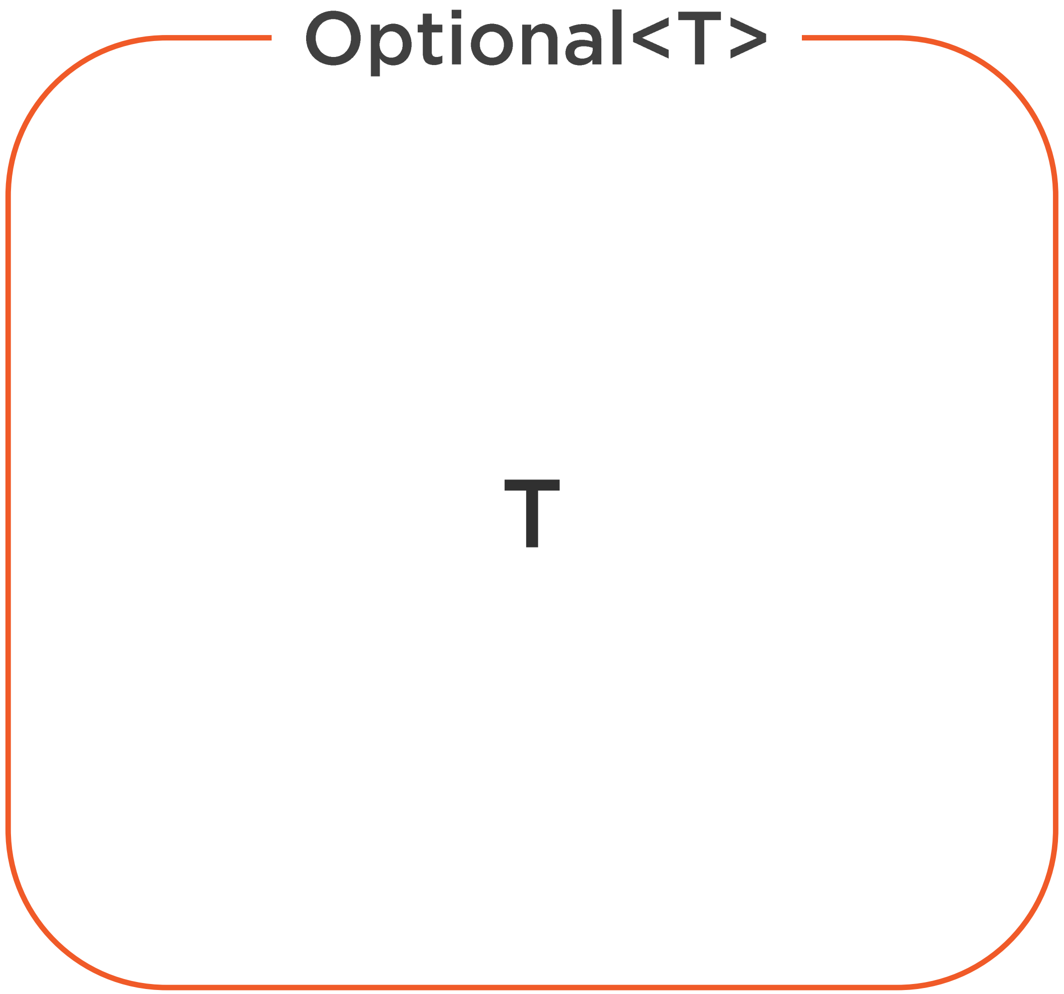
```
Book book = service.findBookById(id);

// ...

String title = book.getTitle();
```

```java
public Book findBookById(int id)
        throws NoBookException {
    // ...
}
```

```java
public List<Book> findBookByAuthor(int authorId) {
    // ...
}
```

List

1, 2, 3, 4, 5


Optional<T>
T

```java
public Optional<Book> findBookById(int id) {
    // ...
}
```

No need to wrap the list in an Optional

```
public List<Book> findBookByAuthor(int authorId) {
    // ...
}
```

Creating an Optional

```
// From a non-null object
Optional<Book> optionalBook = Optional.of(book);

// From an object that may hold a null value
Optional<Book> optionalBook = Optional.ofNullable(book);

// Creating an empty optional
Optional<Book> optionalBook = Optional.empty();
```

Unpacking a Value from an Optional

```
// It can throw a NoSuchElementException
Book book = optionalBook.get();

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

Unpacking a Value from an Optional

```
// To provide a default value
Book book = optionalBook.orElse( new Book() );

// To provide a default value via a Supplier
Book book = optionalBook.orElseGet( () -> new Book() );
```

Methods Similar to the Ones of the Stream API

filter

map

flatMap

Optional Type Good Practices

The best way to use Optional is through composition.

Filter

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

Filter

```
Book book = service.findBookById(id);
if (book != null && book.getNumberOfPages() > 500) {
    System.out.println("It is a long book");
}
```

Filter

```
service.findBookById(id)
    .filter(book -> book.getNumberOfPages() > 500)
    .ifPresent(
        book -> System.out.println("It is a long book")
    );
```

Map

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

Map

```
String title = "";
Book book = service.findBookById(id);
if (book != null){
    title = book.getTitle();
}
```

Map

```
String title = service.findBookById(id)
                      .map(book -> book.getTitle())
                      .orElse("");
```

FlatMap

```
String title = service.findBookById(id)
                      .flatMap(book -> book.getTitle())
                      .orElse("");
```

If getTitle() returns
an Optional, use flatMap

FlatMap

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

If the function returns a plain object, use map.

If the function returns an Optional, use flatMap.

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

Never use Optional.get() unless you're sure that the Optional is not empty

Generally, you shouldn't use Optional in fields

- The Optional class is not serializable
- For truly optional fields, have a getter method return Optional

Don't use Optional as a method argument, it's not necessary

- Use methods of the Optional type itself

Do use optional as a return value

Demo

Using the Optional Type

Things to Keep in Mind About the Optional Type

Optional is a Java class, so a reference of this type might be null

Optional doesn’t address other important sources of NullPointerExceptions

- Partially-initialized objects
- Getting elements from collections or maps

Optional can make your code harder to read

- Types: Optional<Book> vs. Book
- Checking: opt.isPresent() vs. obj != null

Optional can convert a NullPointerException into another exception

Summary

Null is a value that indicates that a reference doesn't refer to an object

- It can appear anywhere causing a NullPointerException

Traditionally, developers use:

- Assertions
- If/else statements
- Methods of the java.util.Objects class
- Try/catch blocks

Summary

For parts of the application where you don’t have control of the data

- Document your public API
- Check for nulls only in the upper layers
- Fail fast
- Use exceptions to indicate that an invalid value has been received

Summary

For parts of the application where you have control of the data

- Never pass null to a method
- Never return null from a method

Summary

Null-safety annotations

- Compile-time
- Run-time

To choose an annotation library consider

- At what point the null check is performed
- Where you can use the annotations
- Tool and language interoperability compatibility

Annotations are not enough

Summary

The Null Object pattern replaces nulls with objects that implement

- A default behavior
- A do-nothing behavior

To implement it:

- Abstract class that defines the behavior for all objects of this type
- The Null Object is a subclass of the abstract class

Summary

The Optional type acts as a container encapsulating either a value of a given type or nothing at all

The best way to use Optional is through composition

- filter
- map
- flatMap

Always start from an optional, apply a chain of methods, and at the end, unwrap the value

Summary

Don’ts

- Don’t use Optional.get()
- Don’t use of Optional as a method argument
- Don’t use Optional in fields unless necessary
    - It’s valid to have a getter method return an Optional

Summary

Things to have in mind about Optional

- A reference of this type might be null
- It doesn't address cases such as partially-initialized objects
- It can make your code harder to read
- You could be swapping one type of exception for another

Thank you