

Avoiding Null



Image: Craig Adderly on pexels.com

Topics

- 1) “My function normally returns an object, but sometimes it cannot”

OK: `myCountry.getBiggestProvinceOrState()`

Problem: `vaticanCity.getBiggestProvinceOrState()`

How can I make this less painful?

- 2) “My object’s field can reference null”

OK: `Cellphone` has-a `SimCard`

Problem: but it might not have one!

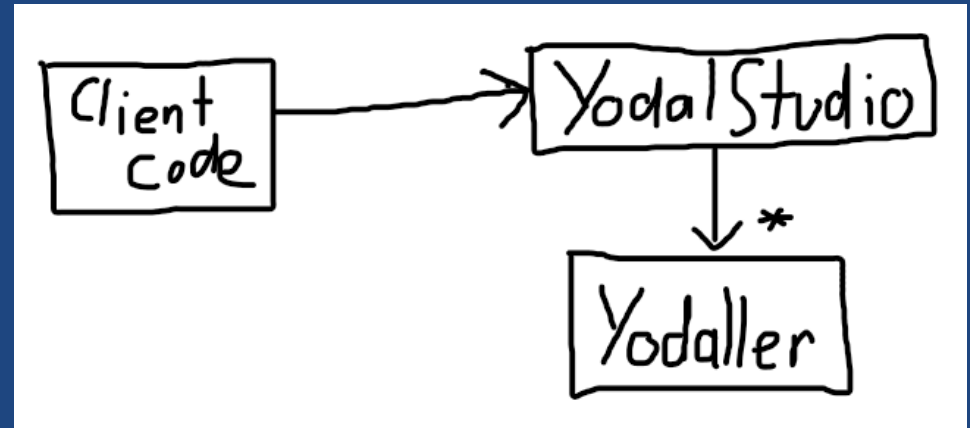
How can I remove all my null reference checks?

Avoid Returning Null



Motivation

- **Problem:**
Sometimes a function cannot return a valid object
- **Example:**
 - **Client** wants to print the name of the loudest **Yodaller**
 - **Client** asks **YodaStudio** for the loudest **Yodaller**
- **What's the challenge?**



Idea 1: Return null

```
public Yodaller getLoudest_1() {
    if (students.isEmpty()) {
        return null;
    }

    Yodaller loudest = null;
    for (Yodaller y : students) {
        if (loudest == null ||
            y.getVolume() > loudest.getVolume())
        {
            loudest = y;
        }
    }
    return loudest;
}
```

```
void printLoudestName(YodaStudio studio) {
    Yodaller loudest = studio.getLoudest_1();
    if (loudest != null) {
        System.out.println(loudest.getName());
    } else {
        System.out.println("Nobody");
    }
}
```

Good

- null is fast and easy to code

Bad

- ..
- client code must remember to always check for null

- ..

Idea 2: Throw checked exception

```
class NoEntries  
    extends Exception  
    {}
```

```
public Yodaller getLoudest_2() throws NoEntries {  
    if (students.isEmpty()) {  
        throw new NoEntries();  
    }  
  
    Yodaller loudest = null;  
    for (Yodaller y : students) {  
        if (loudest == null ||  
            y.getVolume() > loudest.getVolume())  
        {  
            loudest = y;  
        }  
    }  
    return loudest;  
}
```

Good

– ..

Bad

- Exceptions should be for exceptional events; not expected edge cases
- try-catch code breaks standard logic flow, is hard to read
- Exceptions can be expensive to create

```
void printLoudestName(YodalStudio studio) {  
    try {  
        Yodaller loudest = studio.getLoudest_2();  
        System.out.println(loudest.getName());  
    } catch (NoEntries e) {  
        System.out.println("Nobody");  
    }  
}
```

Idea 3: Optional

- Optional

..

- Suggested Use

Use Optional for methods which (both)

1) cannot always return a value, and

2) client code should *always* check for a valid value

Idea 3: Optional (cont)

```
public Optional<Yodaller> getLoudest_3() {
    if (students.isEmpty()) {
        return Optional.empty();
    }

    Yodaller loudest = null;
    for (Yodaller y : students) {
        if (loudest == null ||
            y.getVolume() > loudest.getVolume())
        {
            loudest = y;
        }
    }
    return Optional.of(loudest);
}
```

Good

– ..

- Convenient methods to minimize calling code logic

```
void printLoudestName(YodalStudio studio) {
    Optional<Yodaller> loudest = studio.getLoudest_3();
    String name = loudest.map(s -> s.getName()).orElse("Nobody");
    System.out.println(name);
}
```


Creating an Optional

- **Optional.empty()**

Gives an "empty" (not containing a reference) **Optional**

- **Optional.of(myObj)**

Holds reference to **myObj**; throws **NullPointerException** if null

```
Optional<String> getLastCommand() {  
    List<String> data = getCommandHistory();  
    if (data.isEmpty()) {  
        return Optional.empty();  
    }  
  
    return Optional.of(data.get(data.size() - 1));  
}
```

- **Optional.ofNullable(myObj)**

sets it to be either empty (if null), or contain object

Handling an Optional: `orElse()`

- `myOptional.orElse(someDefault)`

..

```
void printLastCommand() {  
    Optional<String> lastCommand = getLastCommand();  
  
    String msg = lastCommand.orElse("No commands yet");  
  
    System.out.println("Last command: " + msg);  
}
```

Handling an Optional: `map()`

- Use `map()` to work with an `Optional` in powerful ways
- Give it an argument of a lambda function (a "consumer") to process the contained object (if any).

```
void printLoudestName_3(YodaStudio studio) {  
    Optional<Yodaller> loudest = studio.getLoudest_3();  
    String name = loudest  
        .map(s -> s.getName())  
        .orElse("Nobody");  
    System.out.println(name);  
}
```

- Can do more complex operations

```
String s = loudest  
    .map(s -> myFormatter.formatName(s.getName()))  
    .orElse("Nobody");
```

Handling an Optional: Direct access

- Can manually check state and then access
 - use `isPresent()` or `isEmpty()` to check state
 - use `get()` to access element
(throws `NoSuchElementException` if empty)

```
void extraLogic() {
    Optional<String> lastCommand = getLastCommand();
    String msg = "No commands yet";
    if (lastCommand.isPresent()) {
        msg = lastCommand.get();
    }
    System.out.println("Last command: " + msg);
}
```

- Tip: Avoid direct access if you can; it adds extra logic. Use `Optional`'s other methods

**MY FUNCTIONS
DON'T ALWAYS RETURN NULL**



**BUT WHEN THEY
DO, I USE OPTIONAL**

Avoiding has-a Null



Motivation

- When our object has-a reference to an object which could be null, we have to use many null checks.

- **Example**

- **MediaPlayer** holds reference to a **Song** when playing
- Reference set to **null** when no song playing
- **MediaPlayer** must check if **Song** is null on every access:
ex: song duration, song name, artist name,



The null field problem

- **Problems**
 - Code has many checks for null
 - If you miss a null check will crash program instead of doing “nothing”
- **Solution**
 - Instead of referencing a null for no object, reference an object which has the same methods but does “nothing” (or default behaviour)

NullObject

- A **NullObject**
 - ..
(or inherits same base-class) as the normal object,
• ..
- **Code Demo: Patterns-NullObject**
 - See with and without null object
 - Draw class diagram for Null Object pattern

Summary

- Instead of a function returning a null, return an **Optional** object.
 - Simple access to **Optional** with **.orElse()**
 - Power access to **Optional** with **.map()**
- Instead of doing null checks on a possibly null field, use the **Null Object** pattern instead