Working with Nulls in Java

WORKING WITH REFERENCE TYPES AND NULLS

Esteban Herrera
JAVA ARCHITECT

@eh3rrera www.eherrera.net

What exactly is null?

Why is null == null true?

Is it a bad practice to use null?

How can we avoid null checks?

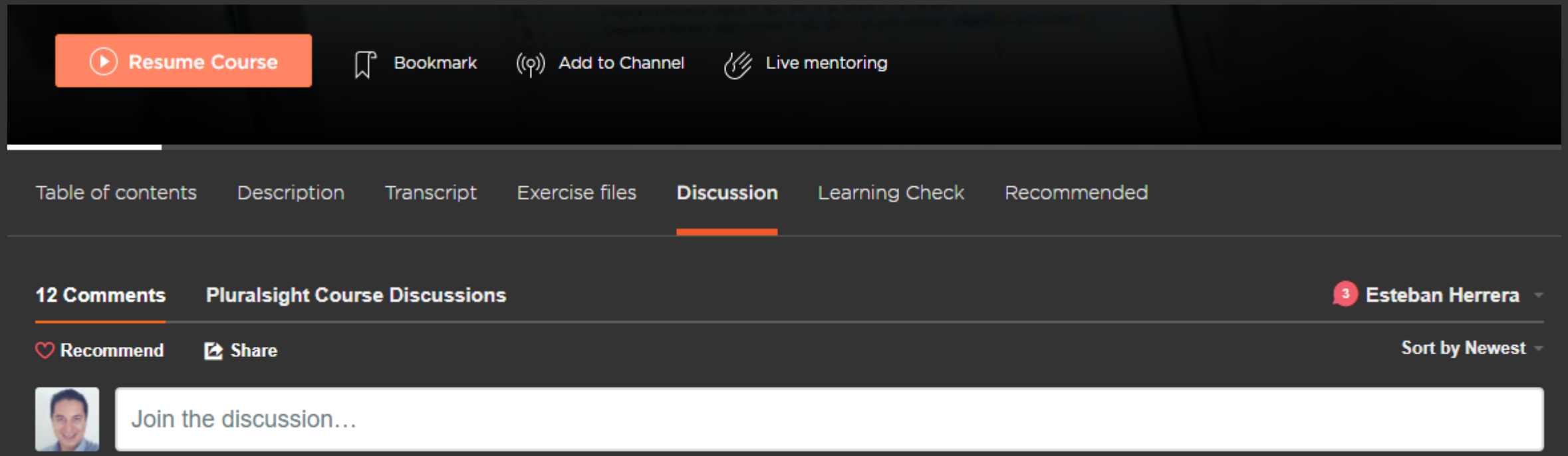
Overview

- Working with reference types and nulls
- Checking for null using annotations
- Using the Null Object pattern
- Using Optional instead of null

Code compatible with Java 8+

During the Course

Ask Questions



Null and Reference Types

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965. [...] 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.”

Sir Charles Antony Richard Hoare

Null Usage

- Optional data
- Unknown data
- Eager deinitialization

Null from a Technical Point of View

```
Object    obj          =          null;
```

Assignment Statement

Assignment

```
Book book = new Book();
```

Reference

Object

Assignment Statement

```
Book  book;
Book  book = null;
```

Wrapper Classes

Primitive Type	Wrapper Class
char	Character
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double
boolean	Boolean

NullPointerException

Thrown when we use a reference that points to nothing instead of an object

A null value can appear anywhere

Nulls cannot be
completely avoided in Java.

Traditional Ways of Dealing with Nulls

Assertions

```
public boolean isBookReadyForPublication(Book book) {
      assert book != null : "Book is null";
      // ...
}
```

```
java -ea com.pluralsight.MyClass com.pluralsight.myPackage...
```

Enabling Assertions

If/Else Statements

```
public boolean isBookReadyForPublication(Book book) {
        if (book != null) {
                // do something with book object
        } else {
                // book object is null
        }
}
```

If/Else Statements

```
public boolean isBookReadyForPublication(Book book) {
        if (null != book) {
                // do something with book object
        } else {
                // book object is null
        }
}
```

java.util.Objects Class

```
public boolean isBookReadyForPublication(Book book) {
      Objects.requireNonNull(book, "Book is null");
      // ...
}
```

java.util.Objects Class

```
public boolean isBookReadyForPublication(Book book) {
      if ( Objects.nonNull(book) ) {
            // do something with book object
      } else {
            // book object is null
      }
}
```

java.util.Objects Class

```
public boolean isBookReadyForPublication(Book book) {
      if ( Objects.isNull(book) ) {
            // book object is null
      } else {
            // do something with book object
      }
}
```

Try/Catch

```
public boolean isBookReadyForPublication(Book book) {
      try {
            // use book object
      } catch(NullPointerException ex) {
            // book object was null
      }
}
```

Best Practices for Data That You Don't Control

Two Types of Data

Outside data

(we don’t control it)

Internal data

(we do control it)

The Problem Is in the Outer Layers

Presentation layer
(or entry point of the app)

Service layer

Persistence layer

Unwanted
null references

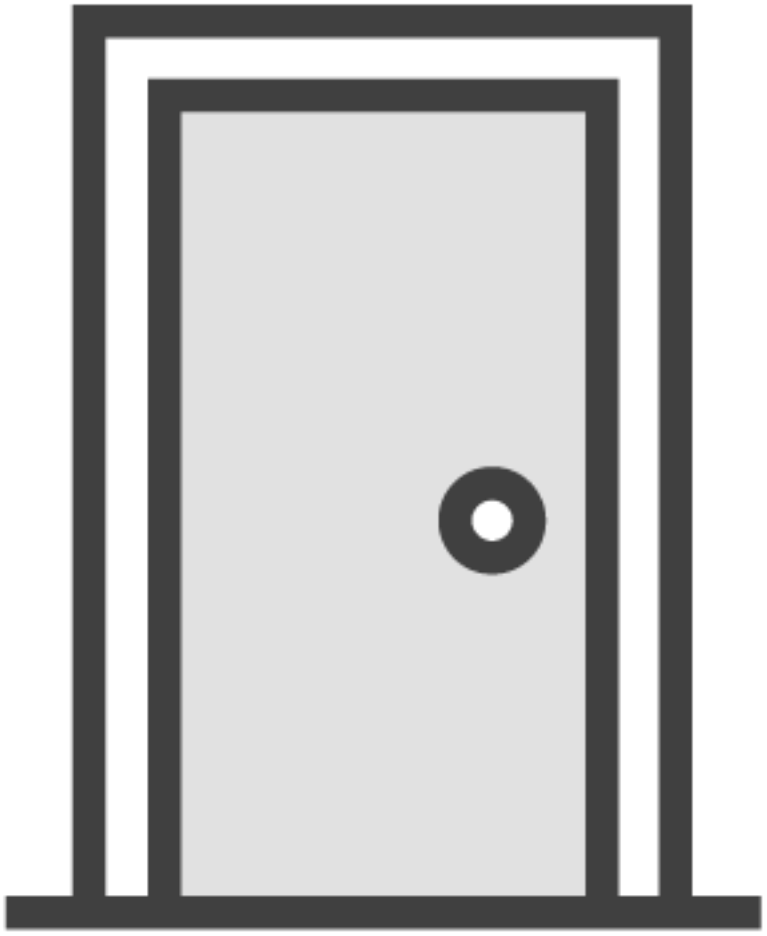
Where to Check for Null?

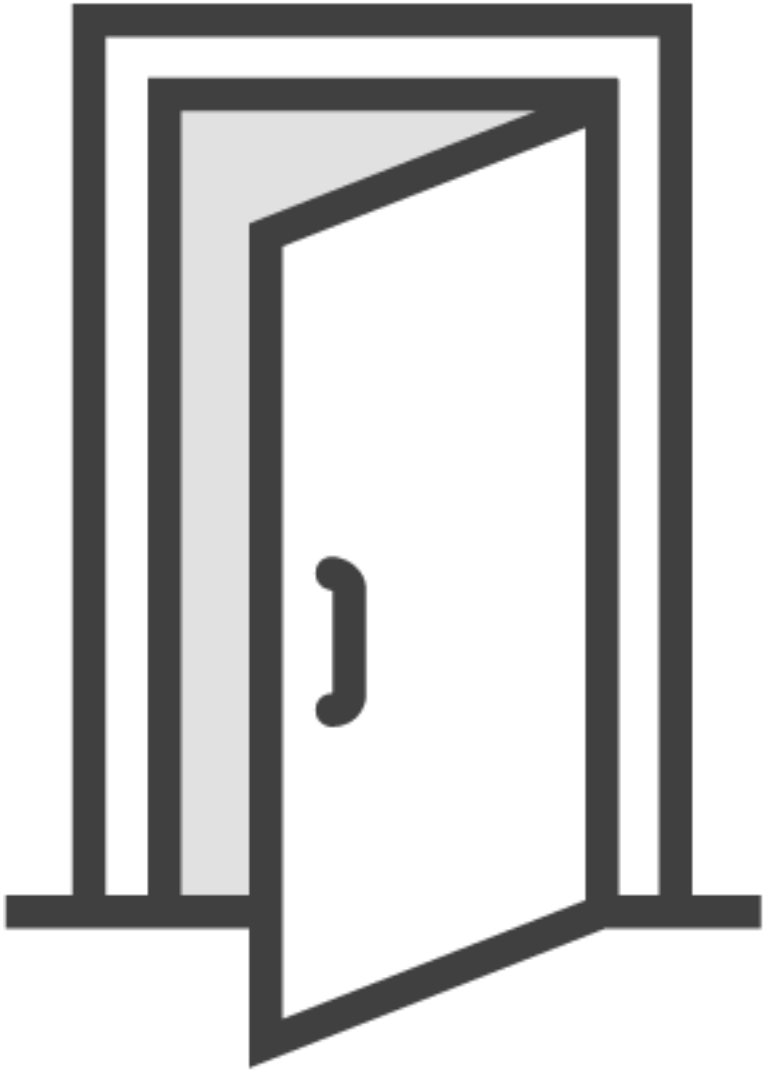
Mostly here

Presentation layer
(or entry point of the app)

Service layer

Persistence layer





For Data You Don't Control

Document public or exposed elements

- Javadoc comments
- For methods describe the contract
  - Preconditions
  - Postconditions
  - Parameters
  - Return values

```
/**
 * Checks if a book is ready for publication, validating:
 * - It has a minimum number of pages
 * - A cover with the right properties
 * If it has an ISBN assigned, it must be valid.
 * This method <i>does not</i> have side-effects.
 *
 * @param  book A book object whose properties will be checked. Must be non-null.
 * @return true if the book is ready for publication, false otherwise.
 * @throws NullPointerException if the book is null.
*/
public boolean isBookReadyForPublication(Book book) {
      // ...
}
```

For Data You Don't Control

Always validate parameters

- At the beginning of the method
  - Fail-fast

For Data You Don't Control

After detecting an invalid null value

- Replace the null value with some default value
  - Not always the right choice
- Throw an exception
  - NullPointerException is the convention

For Data You Don't Control

NullPointerException or IllegalArgumentException?

- Both are RuntimeExceptions
- Both can provide a meaningful message
- Choose one and use it consistently

Isn’t Throwing an Exception Redundant?

```
public boolean isBookReadyForPublication(Book book) {
    Objects.requireNonNull(book, "Book is null");
    Objects.requireNonNull(book.getAuthor(), "Author is null");
    Objects.requireNonNull(book.getTitle(), "Title is null");
    // Business logic of the method...
}
```

Guiding Principles

Fail fast

Be explicit

Best Practices for Data That You Control

For Data You Control

No need to check for null in every method

- Never pass null as an argument
- Never return null

For Data You Control

Never pass null as an argument

- Use primitives instead of wrapper classes
- For optional parameters, you can overload the method with different sets of parameters

Overload Methods

```
public boolean publishBook(Book book, Date publicationDate) {
    // ...
}
```

Overload Methods

```
public boolean publishBook(Book book, Date publicationDate) {
    // ...
}

public boolean publishBook(Book book) {
    // ...
}
```

Never Return Null

```
public HashSet<Edition> getBookEditions() {
    return book.hasEditions()
                ? null
                : new HashSet<Edition>(book.getEditions())
    ;
}
```

Never Return Null

```
public HashSet<Edition> getBookEditions() {
    return book.hasEditions()
            ? new HashSet<Edition>()
            : new HashSet<Edition>(book.getEditions())
    ;
}
```

For Data You Control

Never return null

- Return empty collection
- Null Object pattern
- Optional type

Don’t overcomplicate things

Summary

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

The JLS defines a Null type

- But it cannot be used

The type of the literal value null is Null

- That's why null instanceof Object returns false

Summary

Null can appear anywhere in a Java program

- NullPointerException

To avoid NullPointerExceptions, developers traditionally use:

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

It's better to not overcomplicate things

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

Have a good suite of tests

In the Next Module

Checking for null using annotations