How to Catch and Fix NullPointer Exception in Java

Nov 28, 2025

Table of Contents

Summarize and analyze this article with 👉

🔮 Google AI Mode or 💬 ChatGPT or 🔍 Perplexity or 🤖 Claude or 🐦 Grok (X).

NullPointerException is the most frequently thrown exception in Java applications, accounting for countless crashes.

It occurs when your code tries to use a variable that doesn't point to any object and instead refers to nothing (null).

Think of it like trying to open a door that doesn't exist. You reach for the handle, but there's nothing there—just empty space. Your program does the same thing when it tries to access methods or properties of a null object, and it crashes as a result.

The good news? NullPointerExceptions are highly preventable once you know what to look for.

What Causes NullPointerException

Some of the most common scenarios for a NullPointerException are:

- Calling methods on a null object
- Accessing a null object's properties
- Accessing an index element (like in an array) of a null object
- Passing null parameters to a method
- Incorrect configuration for dependency injection frameworks like Spring
- Using `synchronized` on a null object
- Throwing null from a method that throws an exception

NullPointerException Example

Here is an example of a NullPointerException thrown when the `length()` method of a null `String` object is called:

```
public class NullPointerExceptionExample {
    private static void printLength(String str) {
        System.out.println(str.length()); //This will crash! str is null
    }

    public static void main(String args[]) {
        String myString = null;
        printLength(myString);
    }
}
```

In this example, the `length()` method of a `String` object is called without performing a null check. Since the value of the string passed from the `main()` method is null, running the above code causes a NullPointerException:

```
Exception in thread "main" java.lang.NullPointerException
    at NullPointerExceptionExample.printLength(NullPointerExceptionExample.
    at NullPointerExceptionExample.main(NullPointerExceptionExample.java:8)
```

How to Avoid NullPointerException

The NullPointerException can be avoided using checks and preventive techniques like the following:

- **Check for null before using an object.** You should verify whether an object is null or not before referencing its methods or properties. For example:

```
if (myObject != null) {
System.out.println(myObject.toString()); //Using object if not null
} else {
System.out.println("Object is null");
}
```

- **Using Apache Commons `StringUtils` for String operations**. For example, using `StringUtils.isNotEmpty()` for verifying if a string is not null or empty before using it further.
- **Using primitives rather than objects where possible**. For example, `int` instead of `Integer` and `boolean` instead of `Boolean`. In Java, primitives cannot have null values.
- **Writing methods that return empty objects rather than null where possible**. For example, returning empty collections and empty strings from a method.

Fixing the Example

To fix the NullPointerException in the earlier example, the `String` object should be checked for null or empty values before it is used any further:

```
import org.apache.commons.lang3.StringUtils;

public class NullPointerExceptionExample {
    private static void printLength(String str) {
        if (StringUtils.isNotEmpty(str)) { //Check to ensure not null or em
            System.out.println(str.length());
        } else {
            System.out.println("Empty string");
        }
    }

    public static void main(String args[]) {
        String myString = null;
```

```
        printLength(myString);
    }
}
```

The code here is updated with a check in the `printLength()` method that ensures the string is not null or empty using the apache commons `StringUtils.isNotEmpty()` method. This check avoids the NullPointerException since the `length()` method of the string is called only if it is not null or empty. Otherwise, the message `Empty string` is printed to the console.

Why Prevention Matters

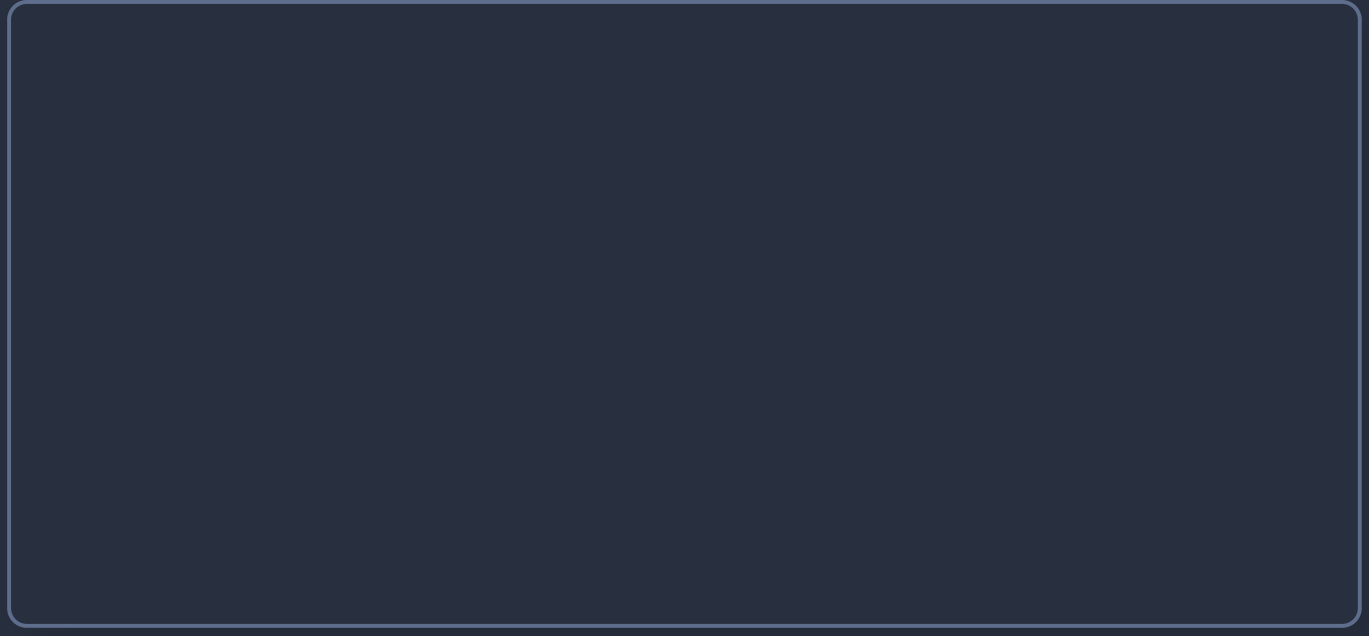
NullPointerExceptions are among the most common runtime errors in production Java applications. A single uncaught NPE can crash user sessions and create poor user experiences. The prevention techniques above, especially defensive null checking, can eliminate the majority of these issues before they reach production. But when exceptions do slip through, having the right monitoring tool makes all the difference.

Track, Analyze and Manage Errors With Rollbar

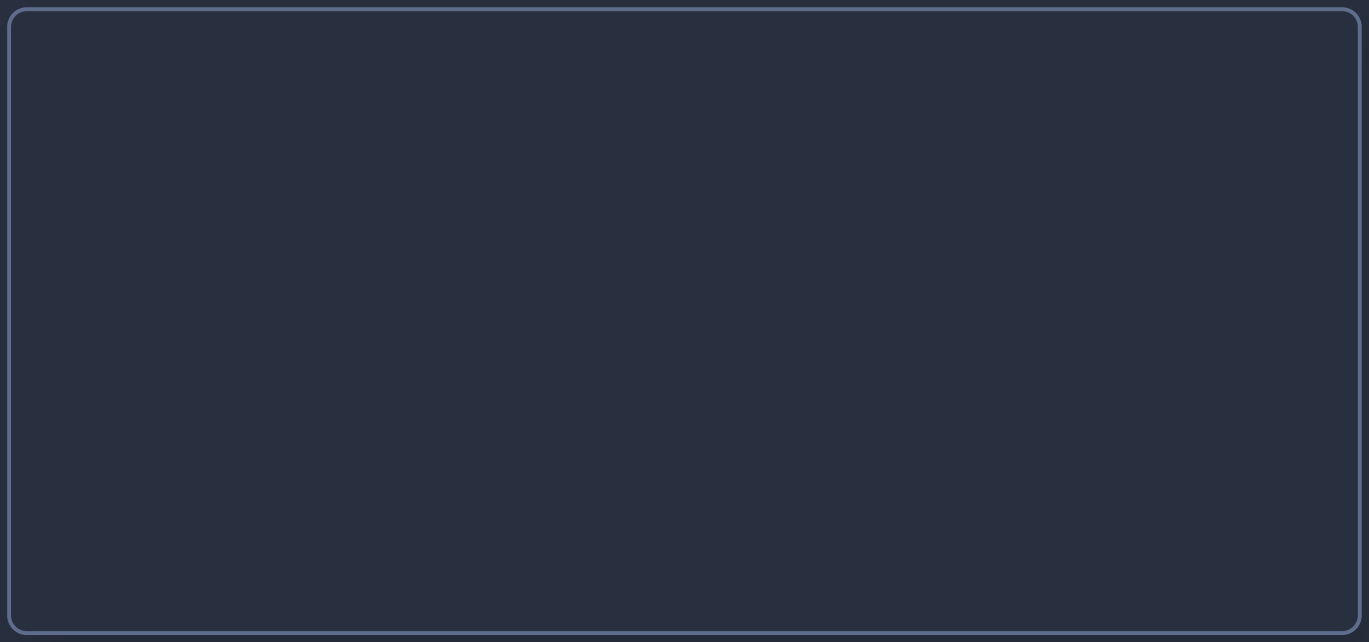
Managing errors and exceptions in your code is challenging. It can make deploying production code an unnerving experience. Being able to track, analyze, and manage errors in real-time can help you to proceed with more confidence. Rollbar automates error monitoring and triaging, making fixing Java errors easier than ever. Try it today!

java

Related Resources



How to Avoid java.util.concurrent.TimeoutException



How to Avoid the Concurrent Modification Exception in Java

How to Debug Java Code Faster with Eclipse

Rollbar

PRODUCT

Product

Pricing

Customers

Platforms

Integrations

Changelog

Roadmap

DOCUMENTATION

Docs Overview

Setting up Rollbar

Notifications

Deploy Tracking

Telemetry

Security & Compliance

API

LANGUAGES/FRAMEWORKS

JavaScript

iOS

Next.js

React

React Native

Rails

Ruby

More...

Go

Java

.NET

Node.js

PHP

Python

Django

COMPANY

About Us

Careers

Contact Us

Github Twitter LinkedIn

Privacy Policy

Terms of Service