Null Pointer Exception in Java

Last Updated : 5 Aug, 2025

A NullPointerException in Java is a RuntimeException. It occurs when a program attempts to use an object reference that has the null value. In Java, "null" is a special value that can be assigned to object references to indicate the absence of a value.

Reasons for Null Pointer Exception

A NullPointerException occurs due to the following reasons:

- Invoking a method from a null object.
- Accessing or modifying a null object’s field.
- Taking the length of null, as if it were an array.
- Accessing or modifying the slots of null objects, as if it were an array.
- Throwing null, as if it were a Throwable value.
- When you try to synchronize over a null object.

Example:

```
public class Geeks {

    public static void main(String[] args) {

        // Reference set to null
        String s = null;

        System.out.println(s.length());
    }
}
```

Output:

```
Hangup (SIGHUP)
Exception in thread "main" java.lang.NullPointerException
        at Geeks.main(Geeks.java:10)
```

Explanation: In this example, the string reference "s" is null. When the program tries to call the length() method, it throws a NullPointerException because there is no actual object.

Why is null Used in Java?

The null value serves as a placeholder and it indicates that no value is assigned to a reference variable. Common applications include:

- **Linked Data Structures**: It represents the end of a list or tree branch.
- **Design Patterns**: This is used in patterns like the Null Object Pattern or Singleton Pattern.

How to Avoid NullPointerException

To avoid the NullPointerException, we must ensure that all the objects are initialized properly, before we use them. When we declare a reference variable, we must verify that object is not null, before we request a method or a field from the objects.

1. Using String Literals in equals()

A very common case problem involves the comparison between a String variable and a literal. The literal may be a String or an element of an Enum. Instead of invoking the method from the null object, consider invoking it from the literal.

Example:

```java
import java.io.*;

class Geeks {
    public static void main (String[] args) {

        // Initializing String variable with null value
        String s = null;

        // Checking if s.equals null
        try
        {
            // This line of code throws NullPointerException because s is null
            if (s.equals("gfg"))
                System.out.print("Same");
            else
                System.out.print("Not Same");
        }
        catch(NullPointerException e)
        {
            System.out.print("NullPointerException Caught");
        }
    }
}
```

Output

```
NullPointerException Caught
```

We can avoid NullPointerException by calling equals on literal rather than object.

```java
import java.io.*;

class Geeks {
    public static void main (String[] args) {

        // Initializing String variable with null value
        String s = null;

        // Checking if s is null using try catch
        try
        {
            if ("gfg".equals(s))
                System.out.print("Same");
            else
                System.out.print("Not Same");
        }
        catch(NullPointerException e)
        {
            System.out.print("Caught NullPointerException");
        }
    }
}
```

Output

```
Not Same
```

Note: Always invoke equals on the literal to avoid calling a method on a null reference.

2. Checking Method Arguments

Before executing the body of the new method, we should first check its arguments for null values and continue with execution of the method, only when the arguments are properly checked. Otherwise, it will throw an IllegalArgumentException and notify the calling method that something is wrong with the passed arguments.

Example:

```java
import java.io.*;

class Geeks {
    public static void main(String[] args) {

        // String s set an empty string and calling getLength()
        String s = "";

        try {
            System.out.println(getLength(s));
        }
        catch (IllegalArgumentException e) {
            System.out.println(
                "IllegalArgumentException caught");
        }

        // String s set to a value and calling getLength()
        s = "GeeksforGeeks";

        try {
            System.out.println(getLength(s));
        }
        catch (IllegalArgumentException e) {
            System.out.println(
                "IllegalArgumentException caught");
        }

        // Setting s as null and calling getLength()
        s = null;

        try {
            System.out.println(getLength(s));
        }
        catch (IllegalArgumentException e) {
            System.out.println(
                "IllegalArgumentException caught");
        }
    }

    public static int getLength(String s)
    {
        if (s == null)
```

```
            throw new IllegalArgumentException(
                "The argument cannot be null");

        return s.length();
    }
}
```

Output

```
0
13
IllegalArgumentException caught
```

3. Use Ternary Operator

The ternary operator can be used to avoid NullPointerException. First, the Boolean expression is evaluated. If the expression is true then, the value1 is returned, otherwise, the value2 is returned. We can use the ternary operator for handling null pointers.

Example:

```java
import java.io.*;

class Geeks {
    public static void main(String[] args)
    {
        String s = null;
        String m = (s == null) ? "" : s.substring(0, 5);

        System.out.println(m);

        s = "Geeksforgeeks";
        m = (s == null) ? "" : s.substring(0, 5);
        System.out.println(m);
    }
}
```

Output

```
Geeks
```

Explanation: The ternary operator helps check for null and avoid operations on null references.

4. Using Optional Class (Java 8+)

In Java 8, Optional class was introduced as a container object which may or may not contain a non-null value. It helps avoid *NullPointerException* by forcing to explicitly handle the case when a value is absent.

Example:

```java
import java.util.Optional;

public class OptionalExample {
    public static void main(String[] args) {
        Optional<String> name = Optional.ofNullable(null);

        // Safe way to access
        System.out.println(name.orElse("Default Name")); // prints: Default Name
    }
}
```

Output

```
Default Name
```

Explination: Optional.ofNullable(value) wraps the value which might be null. orElse() provides a fallback if the value is not present.

Suggested Quiz 3 Questions

What will happen in the following code? Java public class Geeks { public static void main(String[] args) { try { throw new NullPointerException("Demo"); } catch (ArithmeticException e) { System.out.println("Arithmetic Exception"); } finally { System.out.println("Finally block executed"); } } }

A Arithmetic Exception Finally block executed

B NullPointerException Finally block executed

C Finally block executed

D Compilation Error

Login to View Explanation 1/3 < Previous Next >

Comment N nikhil_9856 42

Article Tags: Java Technical Scripter Exception Handling Java-Exceptions

Explore

Java Basics

OOP & Interfaces

Collections

Exception Handling

Java Advanced

Practice Java