Null Pointer Exception In Java

Difficulty Level : Easy Last Updated : 06 Feb, 2019

NullPointerException is a RuntimeException. In Java, a special null value can be assigned to an object reference. NullPointerException is thrown when program attempts to use an object reference that has the null value.

These can be:

- 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 object, as if it were an array.
- Throwing null, as if it were a Throwable value.
- When you try to synchronize over a null object.

Why do we need the null value?

Null is a special value used in Java. It is mainly used to indicate that no value is assigned to a reference variable. One application of null is in implementing data structures like linked list and tree. Other applications include Null Object pattern (See this for details) and Singleton pattern. The Singleton pattern ensures that only one instance of a class is created and also, aims for providing a global point of access to the object.

A sample way to create at most one instance of a class is to declare all its constructors as private and then, create a public method that returns the unique instance of the class:

```
// To use randomUUID function.
import java.util.UUID;
import java.io.*;

class Singleton
{
    // Initializing values of single and ID to null.
    private static Singleton single = null;
    private String ID = null;

    private Singleton()
    {
        /* Make it private, in order to prevent the
           creation of new instances of the Singleton
           class. */
```

```
        // Create a random ID
        ID = UUID.randomUUID().toString();
    }

    public static Singleton getInstance()
    {
        if (single == null)
            single = new Singleton();
        return single;
    }

    public String getID()
    {
        return this.ID;
    }
}

// Driver Code
public class TestSingleton
{
    public static void main(String[] args)
    {
        Singleton s = Singleton.getInstance();
        System.out.println(s.getID());
    }
}
```

Output:

```
10099197-8c2d-4638-9371-e88c820a9af2
```

In above example, a static instance of the singleton class. That instance is initialized at most once inside the Singleton getInstance method.

How to avoid the NullPointerException?

To avoid the NullPointerException, we must ensure that all the objects are initialized properly, before you 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.

Following are the common problems with the solution to overcome that problem.

Case 1 : String comparison with literals

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.

```
// A Java program to demonstrate that invoking a method
// on null causes NullPointerException
import java.io.*;

class GFG
{
    public static void main (String[] args)
    {
        // Initializing String variable with null value
        String ptr = null;

        // Checking if ptr.equals null or works fine.
        try
        {
            // This line of code throws NullPointerException
            // because ptr is null
            if (ptr.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.

```
// A Java program to demonstrate that we can avoid
// NullPointerException
import java.io.*;

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

```
        // Initialing String variable with null value
        String ptr = null;

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

Output:

```
Not Same
```

Case 2 : Keeping a Check on the arguments of a method

Before executing the body of your 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.

```
// A Java program to demonstrate that we should
// check if parameters are null or not before
// using them.
import java.io.*;

class GFG
{
    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");
            }
        }

        // Function to return length of string s. It throws
        // IllegalArgumentException if s is null.
        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
```

Case 3 : Use of 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:

```
// A Java program to demonstrate that we can use
// ternary operator to avoid NullPointerException.
import java.io.*;

class GFG
{
    public static void main (String[] args)
    {
        // Initializing String variable with null value
        String str = null;
        String message = (str == null) ? "" :
                          str.substring(0,5);
        System.out.println(message);

        // Initializing String variable with null value
        str = "Geeksforgeeks";
        message = (str == null) ? "" : str.substring(0,5);
        System.out.println(message);
    }
}
```

Output:

```
Geeks
```

The message variable will be empty if str's reference is **null** as in case 1. Otherwise, if str point to **actual data**, the message will retrieve the first 6 characters of it as in case 2.

Related Article – Interesting facts about Null in Java

This article is contributed by Nikhil Meherwal. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.