**How to avoid NullPointerException in Java**

This may sound like a very silly topic for an experienced Java developer but for novice and intermediate Java developers the most common and trouble causing Java exception is the famous NPE.

It may look very simple but sometimes can get very tricky to avoid for inexperienced Java developers. Following are some of the ways you can avoid a NPE in your next Java project.

**Using Defensive coding**

This is most simplest way to avoid a NPE in your code.

public void printLength(String input) {  
 if (input != null) {  
 System.*out*.println(input.length());  
 }  
}

With defensive code you are simply checking for a null before performing any action on a reference. If it points to null and you try to invoke a method on it, it can throw a NPE. Defensive code avoids this. You can also use the ternary operator to achieve this as well.

System.*out*.println(input != null ? input.length() : 0);

**Using Constants**

This is one of the things I use whenever possible.

public void checkForName(String input) {  
 if (input.equalsIgnoreCase("Shazin")) {  
 System.*out*.println("Shazin is the developer");  
 }  
}

The above code as the potential to throw a NPE if input is pointing to null. This can be rewritten to avoid the NPE as below.

private static final String *NAME* = "Shazin";  
  
public void checkForName(String input) {  
 if (*NAME*.equalsIgnoreCase(input)) {  
 System.*out*.println("Shazin is the developer");  
 }  
}

The above code eliminates the chances of NPE because we are invoking the equalsIgnoreCase method on a constant not the input.

**Using Optional**

Whenever possible it is advised to use Optional. This will also allow to NPE.

public void checkFruit(Optional<String> fruitOptional) {  
 if (fruitOptional.isPresent()) {  
 switch (fruitOptional.get()) {  
 case "APPLE" -> {System.*out*.println("Apple is red color");}  
 case "ORANGE" -> {System.*out*.println("Orange is orange color");}  
 }  
 }  
}

An Optional provides a way to check whether it contains value before accessing the value itself and can be used both as input as well as output from a Java method which makes it a lot more convenient.

**Using Third Party libraries for common operations**

More often than not there will be times when developers have to do a lot of boilerplate code which may involve String, Integer, etc which may point to a null and can cause NPE.

For instance the printLength method from previous example can be rewritten with the use of [Apache Commons Lang](https://commons.apache.org/proper/commons-lang/) library and by using StringUtils.length method.

public void printLength(String input) {  
 System.out.println(StringUtils.length(input));  
}

The boilerplate code to perform defensive code is done in the third party library itself and this can make the code much more concise. Possibly there is already a third party library available which allows you to achieve what you are trying by avoiding a NPE whatever that may be so just try to find it.

**Conclusion**

These are some of the many, many ways to avoid a NPE but most commonly used. If you know any other methods to avoid NPE please mention in the comments. If you like this post please follow me for more similar posts and share this post.