**Poor Error Handling: Program Catches NullPointerException Development Mitigation SOP**

Poor error handling when a program catches a NullPointerException exemplifies bad practices. Programmers typically catch NullPointerExceptions under three circumstances:

1. The program contains a null pointer dereference, catching the resulting exception was easier than fixing the underlying problem
2. The program explicitly throws a NullPointerException to signal an error condition
3. The code is part of a test harness that supplies unexpected input to the classes under test

The last of those circumstances is the only one that is reasonable.

**Defense Against Poor Error Handling: Program Catches NullPointerException**

The program should not deference null pointers and it would be best practice to do a null check before executing actions on a null object to avoid a *NullPointerException*. Using a *NullPointerException* adds more performance overhead than adding null checks. Also, if there are multiple expressions in the try block that could cause a *NullPointerException*, it is more difficult to find out where the error is coming form. Lastly, throwing a *NullPointerException* could cause the program to go into an unusable state.

If the cause of the null pointer deference cannot be eliminated, carefully review the code to make sure that the exception is handled in such a way that the program does not enter into an unexpected or illegal state. Either a *RuntimeException* or *Error* could be thrown instead of explicitly throwing a *NullPointerException*.

**Example**

boolean isName( String s ) {

try {

String names[] = s.split( “ ” );

if( names.length != 2 ) { return false; }

return ( isCapitalized( names[0] ) && isCapitalized( names[1] ) );

} catch ( NullPointerException e ) { return false; }

}

Explanation:

1. The code above checks if a string is a valid name, which requires the input to have two capitalized words separated by one or more spaces
2. If the string is null, the catch block catches a *NullPointerException* and returns false, rather than checking to see if the input is null before performing any other actions in the method
3. The code below shows that it would be better to do a null check (bolded below) before any other actions to avoid throwing a *NullPointerException*:

boolean isName( String s ) {

**if( s == null ) { return false: }**

String names[] = s.split( “ ” );

if( names.length != 2 ) { return false; }

return ( isCapitalized( names[0] ) && isCapitalized( names[1] ) );

}

**References**

1. [HP Enterprise Security – Poor Error Handling: Program Catches NullPointerException](http://www.hpenterprisesecurity.com/vulncat/en/vulncat/java/poor_error_handling_program_catches_nullpointerexception.html)
2. [Software Engineering Institute – ERR08-J](https://www.securecoding.cert.org/confluence/display/java/ERR08-J.+Do+not+catch+NullPointerException+or+any+of+its+ancestors)