EXP01-J. Do not use a null in a case where an object is required

Do not use the `null` value in any instance where an object is required, including the following cases:

- Calling the instance method of a null object
- Accessing or modifying the field of a null object
- Taking the length of `null` as if it were an array
- Accessing or modifying the elements of `null` as if it were an array
- Throwing `null` as if it were a `Throwable` value

Using a `null` in cases where an object is required results in a `NullPointerException` being thrown, which interrupts execution of the program or thread. Code conforming to this coding standard will consequently terminate because ERR08-J. Do not catch NullPointerException or any of its ancestors r equires that `NullPointerException` is not caught.

Noncompliant Code Example

This noncompliant example shows a bug in Tomcat version 4.1.24, initially discovered by Reasoning [Reasoning 2003]. The `cardinality()` method was designed to return the number of occurrences of object `obj` in collection `col`. One valid use of the `cardinality()` method is to determine how many objects in the collection are null. However, because membership in the collection is checked using the expression `obj.equals(elt)`, a null pointer dereference is guaranteed whenever `obj` is null and `elt` is not null.

```
public static int cardinality(Object obj, final Collection<?> col) {
  int count = 0;
  if (col == null) {
    return count;
  }
  Iterator<?> it = col.iterator();
  while (it.hasNext()) {
    Object elt = it.next();
    if ((null == obj && null == elt) || obj.equals(elt)) {  // Null pointer dereference
      count++;
    }
  }
  return count;
}
```

Compliant Solution

This compliant solution eliminates the null pointer dereference by adding an explicit check:

```
public static int cardinality(Object obj, final Collection col) {
  int count = 0;
  if (col == null) {
    return count;
  }
  Iterator it = col.iterator();
  while (it.hasNext()) {
    Object elt = it.next();
    if ((null == obj && null == elt) ||
        (null != obj && obj.equals(elt))) {
      count++;
    }
  }
  return count;
}
```

Noncompliant Code Example

This noncompliant code example defines an `isProperName()` method that returns true if the specified `String` argument is a valid name (two capitalized words separated by one or more spaces):

```
public boolean isProperName(String s) {
  String names[] = s.split(" ");
  if (names.length != 2) {
    return false;
  }
  return (isCapitalized(names[0]) && isCapitalized(names[1]));
}
```

Method `isProperName()` is noncompliant because it may be called with a null argument, resulting in a null pointer dereference.

Compliant Solution (Wrapped Method)

This compliant solution includes the same `isProperName()` method implementation as the previous noncompliant example, but it is now a private method with only one caller in its containing class.

```
public class Foo {
  private boolean isProperName(String s) {
    String names[] = s.split(" ");
    if (names.length != 2) {
      return false;
    }
    return (isCapitalized(names[0]) && isCapitalized(names[1]));
  }

  public boolean testString(String s) {
    if (s == null) return false;
    else return isProperName(s);
  }
}
```

The calling method, `testString()`, guarantees that `isProperName()` is always called with a valid string reference. As a result, the class conforms with this rule even though a public `isProperName()` method would not. Guarantees of this sort can be used to eliminate null pointer dereferences.

Compliant Solution (Optional Type)

This compliant solution uses an `Optional String` instead of a `String` object that may be null. The `Optional` class (`java.util.Optional` [API 2014]) was introduced in Java 8 and can be used to mitigate against null pointer dereferences.

```
public boolean isProperName(Optional<String> os) {
  String names[] = os.orElse("").split(" ");
  return (names.length != 2) ? false :
         (isCapitalized(names[0]) && isCapitalized(names[1]));
}
```

The `Optional` class contains methods that can be used to make programs shorter and more intuitive [Urma 2014].

Exceptions

EXP01-J-EX0: A method may dereference an object-typed parameter without guarantee that it is a valid object reference provided that the method documents that it (potentially) throws a `NullPointerException`, either via the `throws` clause of the method or in the method comments. However, this exception should be relied on sparingly.

Risk Assessment

Dereferencing a null pointer can lead to a denial of service. In multithreaded programs, null pointer dereferences can violate cache coherency policies and can cause resource leaks.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP01-J	Low	Likely	High	**P3**	**L3**

Automated Detection

Null pointer dereferences can happen in path-dependent ways. Limitations of automatic detection tools can require manual inspection of code [Hovemeyer 2007] to detect instances of null pointer dereferences. Annotations for method parameters that must be non-null can reduce the need for manual inspection by assisting automated null pointer dereference detection; use of these annotations is strongly encouraged.

Tool	Version	Checker	Description
The Checker Framework	2.1.3	**Nullness Checker** **Initialization Checker** **Map Key Checker**	Null pointer errors (see Chapter 3) Ensure all fields are set in the constructor (see Chapter 3.8) Track which values are keys in a map (see Chapter 4)
CodeSonar	4.2	**FB.CORRECTNESS.NP_ALWAYS_NULL** **FB.CORRECTNESS.NP_ALWAYS_NULL_EXCEPTION** **FB.CORRECTNESS.NP_ARGUMENT_MIGHT_BE_NULL** **FB.BAD_PRACTICE.NP_BOOLEAN_RETURN_NULL** **FB.BAD_PRACTICE.NP_CLONE_COULD_RETURN_NULL** **FB.CORRECTNESS.NP_CLOSING_NULL** **FB.STYLE.NP_DEREFERENCE_OF_READLINE_VALUE** **FB.BAD_PRACTICE.** **NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT** **FB.CORRECTNESS.NP_GUARANTEED_DEREF** **FB.CORRECTNESS.** **NP_GUARANTEED_DEREF_ON_EXCEPTION_PATH** **FB.STYLE.NP_IMMEDIATE_DEREFERENCE_OF_READLINE** **FB.STYLE.NP_LOAD_OF_KNOWN_NULL_VALUE** **FB.CORRECTNESS.** **NP_NONNULL_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR** **FB.CORRECTNESS.NP_NONNULL_PARAM_VIOLATION** **FB.CORRECTNESS.NP_NONNULL_RETURN_VIOLATION** **FB.STYLE.NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE** **FB.CORRECTNESS.NP_NULL_ON_SOME_PATH_EXCEPTION** **FB.STYLE.NP_NULL_ON_SOME_PATH_MIGHT_BE_INFEASIBLE** **FB.CORRECTNESS.NP_NULL_ON_SOME_PATH** **FB.CORRECTNESS.NP_NULL_PARAM_DEREF** **FB.CORRECTNESS.NP_NULL_PARAM_DEREF_NONVIRTUAL** **FB.CORRECTNESS.** **NP_NULL_PARAM_DEREF_ALL_TARGETS_DANGEROUS** **FB.STYLE.** **NP_PARAMETER_MUST_BE_NONNULL_BUT_MARKED_AS_NULLA** **BLE** **FB.CORRECTNESS.NP_STORE_INTO_NONNULL_FIELD** **FB.CORRECTNESS.NP_UNWRITTEN_FIELD** **FB.STYLE.NP_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD** **FB.CORRECTNESS.** **RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE** **FB.BAD_PRACTICE.NP_TOSTRING_COULD_RETURN_NULL**	Null pointer dereference Null pointer dereference in method on exception path Method does not check for null argument Method with Boolean return type returns explicit null Clone method may return null close() invoked on a value that is always null Dereference of the result of readLine() without nullcheck equals() method does not check for null argument Null value is guaranteed to be dereferenced Value is null and guaranteed to be dereferenced on exception path Immediate dereference of the result of readLine() Load of known null value Non-null field is not initialized Method call passes null to a non-null parameter Method may return null, but is declared @Nonnull Possible null pointer dereference due to return value of called method Possible null pointer dereference in method on exception path Possible null pointer dereference on branch that might be infeasible Possible null pointer dereference Method call passes null for non-null parameter (deref) Non-virtual method call passes null for non-null parameter Method call passes null for non-null parameter (deref all) Parameter must be non-null but is marked as nullable Store of null value into field annotated @Nonnull Read of unwritten field Read of unwritten public or protected field Nullcheck of value previously dereferenced toString method may return null

Coverity	v7.5	**FORWARD_NULL NULL_RETURNS REVERSE_INULL FB.BC_NULL_INSTANCEOF FB.NP_ALWAYS_NULL FB.NP_ALWAYS_NULL_EXCEPTION FB.NP_ARGUMENT_MIGHT_BE_NULL FB.NP_BOOLEAN_RETURN_NULL FB.NP_CLONE_COULD_RETURN_NULL FB.NP_CLOSING_NULL FB.NP_DEREFERENCE_OF_ READLINE_VALUE FB.NP_DOES_NOT_HANDLE_NULL FB.NP_EQUALS_SHOULD_HANDLE_ NULL_ARGUMENT FB.NP_FIELD_NOT_INITIALIZED_ IN_CONSTRUCTOR FB.NP_GUARANTEED_DEREF FB.NP_GUARANTEED_DEREF_ON_ EXCEPTION_PATH FB.NP_IMMEDIATE_DEREFERENCE_ OF_READLINE FB.NP_LOAD_OF_KNOWN_NULL_ VALUE FB.NP_NONNULL_FIELD_NOT_ INITIALIZED_IN_CONSTRUCTOR FB.NP_NONNULL_PARAM_VIOLATION FB.NP_NONNULL_RETURN_VIOLATION FB.NP_NULL_INSTANCEOF FB.NP_NULL_ON_SOME_PATH FB.NP_NULL_ON_SOME_PATH_ EXCEPTION FB.NP_NULL_ON_SOME_PATH_ FROM_RETURN_VALUE FB.NP_NULL_ON_SOME_PATH_ MIGHT_BE_INFEASIBLE FB.NP_NULL_PARAM_DEREF FB.NP_NULL_PARAM_DEREF_ALL_ TARGETS_DANGEROUS FB.NP_NULL_PARAM_DEREF_ NONVIRTUAL FB.NP_PARAMETER_MUST_BE_NON - NULL_BUT_MARKED_AS_NULLABLE FB.NP_STORE_INTO_NONNULL_FIELD FB.NP_TOSTRING_COULD_ RETURN_NULL FB.NP_UNWRITTEN_FIELD FB.NP_UNWRITTEN_PUBLIC_OR_ PROTECTED_FIELD FB.RCN_REDUNDANT_COMPARISON_ OF_NULL_AND_NONNULL_VALUE FB.RCN_REDUNDANT_COMPARISON_ TWO_NULL_VALUES FB.RCN_REDUNDANT_NULLCHECK_ OF_NONNULL_VALUE FB.RCN_REDUNDANT_NULLCHECK_ OF_NULL_VALUE FB.RCN_REDUNDANT_NULLCHECK_ WOULD_HAVE_BEEN_A_NPE**	Implemented
Fortify	V. 5.0	**Missing_Check_against_Null Null_Dereference Redundant_Null_Check**	Implemented
Findbugs	V. 2.0	**NP_DEREFERENCE_OF_READLINE_VALUE NP_NULL_PARAM_DEREF NP_TOSTRING_COULD_RETURN_NULL**	Implemented
Parasoft Jtest	10.3	**BD.EXCEPT.NP, PB-RE-NMCD**	
SonarQube	6.7	**S2259 S2225 S2447 S2637**	Null pointers should not be dereferenced "toString()" and "clone()" methods should not return null Null should not be returned from a "Boolean" method "@NonNull" values should not be set to null

Related Vulnerabilities

Java Web Start applications and applets particular to JDK version 1.6, prior to update 4, were affected by a bug that had some noteworthy security consequences. In some isolated cases, the application or applet's attempt to establish an HTTPS connection with a server generated a `NullPointerException` [SDN 2008]. The resulting failure to establish a secure HTTPS connection with the server caused a denial of service. Clients were temporarily forced to use an insecure HTTP channel for data exchange.

Related Guidelines

SEI CERT C Coding Standard	EXP34-C. Do not dereference null pointers
ISO/IEC TR 24772:2010	Null Pointer Dereference [XYH]
MITRE CWE	CWE-476, NULL Pointer Dereference

Android Implementation Details

Android applications are more sensitive to `NullPointerException` because of the constraint of the limited mobile device memory. Static members or members of an Activity may become null when memory runs out.

Bibliography

[API 2006]	Method `doPrivileged()`
[API 2014]	Class `java.util.Optional`
[Hovemeyer 2007]	
[Reasoning 2003]	"Defect ID 00-0001" "Null Pointer Dereference"
[SDN 2008]	Bug ID 6514454
[Seacord 2015]	EXP01-J. Never dereference null pointers LiveLesson
[Urma 2014]	Tired of Null Pointer Exceptions? Consider Using Java SE 8's Optional!
