ERR07-J. Do not throw RuntimeException, Exception, or Throwable

Methods must not throw `RuntimeException`, `Exception`, or `Throwable`. Handling these exceptions requires catching `RuntimeException`, which is disallowed by ERR08-J. Do not catch NullPointerException or any of its ancestors. Moreover, throwing a `RuntimeException` can lead to subtle errors; for example, a caller cannot examine the exception to determine why it was thrown and consequently cannot attempt recovery.

Methods can throw a specific exception subclassed from `Exception` or `RuntimeException`. Note that it is permissible to construct an exception class specifically for a single `throw` statement.

Noncompliant Code Example

The `isCapitalized()` method in this noncompliant code example accepts a string and returns true when the string consists of a capital letter followed by lowercase letters. The method also throws a `RuntimeException` when passed a null string argument.

```
boolean isCapitalized(String s) {
  if (s == null) {
    throw new RuntimeException("Null String");
  }
  if (s.equals("")) {
    return true;
  }
  String first = s.substring(0, 1);
  String rest = s.substring(1);
  return (first.equals(first.toUpperCase()) &&
          rest.equals(rest.toLowerCase()));
}
```

A calling method must also violate ERR08-J. Do not catch NullPointerException or any of its ancestors to determine whether the `RuntimeException` was thrown.

Compliant Solution

This compliant solution throws `NullPointerException` to denote the specific exceptional condition:

```
boolean isCapitalized(String s) {
  if (s == null) {
    throw new NullPointerException();
  }
  if (s.equals("")) {
    return true;
  }
  String first = s.substring(0, 1);
  String rest = s.substring(1);
  return (first.equals(first.toUpperCase()) &&
          rest.equals(rest.toLowerCase()));
}
```

Note that the null check is redundant; if it were removed, the subsequent call to `s.equals("")` would throw a `NullPointerException` when `s` is null. However, the null check explicitly indicates the programmer's intent. More complex code may require explicit testing of invariants and appropriate `throw` statements.

Noncompliant Code Example

This noncompliant code example specifies the `Exception` class in the `throws` clause of the method declaration for the `doSomething()` method:

```
private void doSomething() throws Exception {
  //...
}
```

Compliant Solution

This compliant solution declares a more specific exception class in the `throws` clause of the method declaration for the `doSomething()` method:

```
private void doSomething() throws IOException {
  //...
}
```

Exceptions

ERR07-J-EX0: Classes that sanitize exceptions to comply with a security policy are permitted to translate specific exceptions into more general exceptions. This translation could potentially result in throwing `RuntimeException`, `Exception`, or `Throwable` in some cases, depending on the requirements of the security policy.

Risk Assessment

Throwing `RuntimeException`, `Exception`, or `Throwable` prevents classes from catching the intended exceptions without catching other unintended exceptions as well.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ERR07-J	Low	Likely	Medium	P6	L2

Automated Detection

Tool	Version	Checker	Description
Parasoft Jtest	10.3	**CODSTD.BP.NTX, EXCEPT.NTERR**	Implemented
SonarQube	6.7	**S112**	Generic exceptions should never be thrown

Related Guidelines

MITRE CWE	CWE-397, Declaration of Throws for Generic Exception

Bibliography

[Goetz 2004b]	
[Java Tutorials]	Unchecked Exceptions—The Controversy