rules.sonarsource.com

Kotlin static code analysis: Jump statements should not occur in "finally" blocks

3 minutes

Using `break`, `continue`, `return` and `throw` inside of a `finally` block suppresses the propagation of any unhandled `Throwable` thrown in the `try` or `catch` block.

This rule raises an issue when a jump statement (`break`, `continue`, `return`, `throw`) would force control flow to leave a `finally` block.

Noncompliant Code Example

```
fun main() {
    try {
        doSomethingWhichThrowsException(5)
        println("OK") // incorrect "OK" message is printed
    } catch (e: RuntimeException) {
        println("ERROR") // this message is not shown
    }
    try {
        doSomethingThatAlsoThrowsException(5)
        println("OK") // incorrect "OK" message is printed
    } catch (e: RuntimeException) {
        println("ERROR") // this message is not shown
    }
}

fun doSomethingWhichThrowsException(q: Int) {
    try {
        throw RuntimeException()
    } finally {
        //...
        if (someOtherCondition) {
            return  // Noncompliant - prevents the RuntimeException
from being propagated
        }
        if (aLastConditionIsVerified) {
            throw IllegalStateException()  // Noncompliant - prevents the
RuntimeException from being propagated
        }
    }
}

fun doSomethingThatAlsoThrowsException(q: Int) {
    while (someConditionIsVerified) {
        try {
            throw RuntimeException()
        } finally {
            //...
            if (someOtherCondition) {
                continue  // Noncompliant - prevents the
RuntimeException from being propagated
            }
            break  // Noncompliant - prevents the RuntimeException
from being propagated
        }
    }
}
```

Compliant Solution

```
fun main() {
    try {
        doSomethingWhichThrowsException()
        println("OK")
    } catch (e: RuntimeException) {
        println("ERROR") // prints "ERROR" as expected
    }
}
```

```
fun doSomethingWhichThrowsException(q: Int) {
    try {
        throw RuntimeException()
    } finally {
        while (someConditionIsVerified) {
            //...
            if (someOtherCondition) {
                continue  // Compliant - does not prevent the
RuntimeException from being propagated

            }
            break  // compliant - does not prevent the RuntimeException
from being propagated
        }
    }
}
```