[**Remove the declaration of thrown exception 'java.lang.Exception', as it cannot be thrown from method's body.**](http://localhost:9010/project/issues?inNewCodePeriod=true&issueStatuses=OPEN%2CCONFIRMED&open=dd0a1016-a379-416d-b051-f01847b1b991&id=gcis_ss_client)

A screenshot of a computer

Description automatically generated

A close-up of a message

Description automatically generated

A screenshot of a computer

Description automatically generated

**Exceptions**

The rule will not raise any issue for exceptions that cannot be thrown from the method body:

* in interface default methods
* in overriding and implementating methods
* in non-private methods that only throw, have empty bodies, or a single return statement.
* in overridable methods (non-final, or not member of a final class, non-static, non-private), if the exception is documented with a proper JavaDoc

interface MyInterface {

default void defaultMethod() throws IOException {

System.out.println("Hi!");

}

void doSomething() throws IOException;

}

class A implements MyInterface {

@Override

void doSomething() throws IOException {

System.out.println("Hi!");

}

public void emptyBody() throws IOException {}

protected void singleThrowStatement() throws IOException {

throw new UnsupportedOperationException("This method should be implemented in subclasses");

}

Object singleReturnStatement() throws IOException {

return null;

}

/\*\*

\* @throws IOException Overriding classes may throw this exception if they print values into a file

\*/

protected void overridable() throws IOException { // no issue, method is overridable and the exception has proper javadoc

System.out.println("foo");

}

}

Also, the rule will not raise issues on RuntimeException, or one of its sub-classes, because documenting runtime exceptions which could be thrown can ultimately help users of the method understand its behavior.

class B {

int possibleDivisionByZero(int a, int b) throws ArithmeticException {

return a / b;

}

}