

ExceptionInjector

The [ExceptionInjector](#) is a listener that can throw user configured exceptions at arbitrary program locations. The main purpose is to ease the testing of exception handler code that would otherwise hard to reach, e.g. because it is not clear if/how an exception could be thrown in 3rd party code.

Properties

ei.exception = <exception-spec>;...

```
<exception-spec> := <type>'@'<location>
<type> := <exception-classname>[<string-literal? '>']
<location> := <classname>:'line | <classname>.'<method-spec>[:'line]
```

Relative line numbers count from the first executable statement in the method body. They are mostly used to have tests that are more robust against changes in the target source file.

If a method is given without a line offset, the exception is thrown before executing INVOKE instructions that call the specified method.

If a line is specified (either absolute or method-body relative), the exception is thrown before executing the associated bytecode at this line.

If more than one exception specification is given, these need to be separated by semicolons ';' (commas cannot be used because they can appear within argument type lists of the target method).

Method argument types have to be specified the same way as they are reported by 'javap', i.e. with fully qualified type names in dot notation (e.g. "foo(java.lang.String,int[])"). Return types can be omitted.

ei.throw_first [boolean] - if true, throw exception on first bytecode instruction associated with the given (absolute or relative) source line. If false, throw on last associated bytecode instruction

Examples

(1) throw exception on absolute line 42 in xyz.MyClass:

The application property file

```
target = xyz.MyClass
ei.exception = ArithmeticException@xyz.MyClass:42
```

on file

```
1: package x.y.z;
..
public class MyClass {
..
    try {
..
42:     int z = x / y;
..
    } catch (ArithmeticException ax){
        // the handler code to test
    }
..
```

will throw an ArithmeticException on line 42 regardless of the 'x' and 'y' values.

(2) throw a Zapp("gotcha") exception on (relative) line 2 in method body xyz.MyClass.foo()

The application property file

```
target = xyz.MyClass
ei.exception = Zapp("gotcha")@xyz.MyClass.foo(int[]):2
```

on file

```
package x.y.z;
..
public class MyClass {
    ..
    void foo (int[] whatever) {
        // some comment (doesn't count for line offsets)
+0:     int n = ..          // first statement line in foo()
+1:     // some more comment (does count for line offsets)
+2:     doSomething(n); // can throw a Zapp exception
    ..
}
```

will throw a Zapp("gotcha") exception on relative line 2 of method body xyz.MyClass.foo(). Note that the line offset counts from the first executable statement line within foo().

(3) throw an IOException when calling File.createTempFile()

The application property file

```
target = xyz.MyClass
ei.exception = java.io.IOException@java.io.File.createTempFile(java.lang.String,java.lang.String)
```

will throw an exception on the first call of File.createTempFile(), regardless of where this occurs and what the parameters to the call are

```
1: package x.y.z;
..
public class MyClass {
    ..
    try {
        ..
        File tmp = File.createTempFile(prefix,suffix);
        ..
    } catch (IOException iox) {
        // the handler code to test
    }
    ..
}
```