Nolix

Nolix tests

2018-09-20

Table of contents

1 Tests

1.1 What, why and how

What Nolix tests are

Nolix test classes are base classes of custom tests. The Nolix tests use the following terminology.

test unit	An object whose creation or methods are tested.
test case / test method	A method that tests the creation or a method of a test unit.
test / test class	A class whose purpose is, and only is, to contain test cases. The test units of the test cases of a test should be all of the same type.
unit test / unit test class	In general, the dependencies and underlying functionalities the test units use are extracted away and should explicitly not influence the result of a unit test.
Integration test / integration test class	In general, the dependencies and underlying functionalities the test units use are not extracted away and should explicitly influence the result of an integration test.

Why to use Nolix tests

- Nolix tests provide **various** methods to validate test units.
- Validations of test units can be written in **very legible** code.

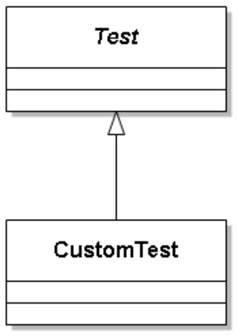
How to use Nolix tests

To use the Nolix tests, import the Nolix library into your project. Then you can access the Nolix tests in your project. The Nolix tests can be found in the package ‘ch.nolix.primitive.test2’.

1.2 How to define a custom test

A custom test can be defined by inheriting from the test class. A test is empty at the beginning.

Example



```
import ch.nolix.core.test2.test;

public class CustomTest extends Test {

}
```

1.3 How to add test cases to a test

To a test, an arbitrary number of test cases can be added. A test case always must be public. All public methods are test cases.

Example

Lets add a first test case to our custom test. This test case tests the length method of a string.

```
import ch.nolix.core.test2.Test;

public class CustomTest extends Test {

     //test case
     public void test_length() {

           //setup
           var testUnit = “Hello!”;

           //execution
           var length = testUnit.length();

           //verification
           expect(length).isEqualTo(6);
     }
}
```

The output of the run of this test is:

```
   PASSED: testLength (0ms)
 = CustomTest: 1/1 test cases passed (0ms)
```

1.4 How to create and run a test

To run a test, create an instance of it and call the method ‘run’ on it.

Example

```
var test = new CustomTest();
test.run();
```

This can be done in 1 line.

```
new CustomTest().run();
```

The run method runs all test cases of a test for 1 time. The test cases of a test are run in alphabetic order.

1.5 What the output of a test is

When a test is run, the following output is produced:

- The running duration of all test cases.
- The number of the passed test cases.
- The total running duration of the test.
- All errors of all failed test cases.

The running duration is always shown in milliseconds (ms).

Example

The following is the output of a test of a Matrix class. One test case of the test failed.

```
   PASSED: loopTest_createIdendityMatrix (0ms)
   PASSED: loopTest_getRank (0ms)
 ->FAILED: loopTest_getTrace: (0ms)
   #1 1.0 was expected, but 0.0 was received.
   (ch.nolix.coreTest.mathematicsTest.MatrixTest.java:69)
   #2 2.0 was expected, but 0.0 was received.
   (ch.nolix.coreTest.mathematicsTest.MatrixTest.java:69)
   #3 3.0 was expected, but 0.0 was received.
   (ch.nolix.coreTest.mathematicsTest.MatrixTest.java:69)
   #4 4.0 was expected, but 0.0 was received.
   (ch.nolix.coreTest.mathematicsTest.MatrixTest.java:69)
   #5 5.0 was expected, but 0.0 was received.
   (ch.nolix.coreTest.mathematicsTest.MatrixTest.java:69)
   PASSED: test_add (0ms)
   PASSED: test_appendAtRight (0ms)
   PASSED: test_getInverse (0ms)
   PASSED: test_getInverse_2 (0ms)
   PASSED: test_getInverse_3 (0ms)
   PASSED: test_getProduct (0ms)
   PASSED: test_getSolutionAsExtendedMatrix (0ms)
   PASSED: test_getSolutionAsExtendedMatrix_2 (0ms)
   PASSED: test_getTransposed (0ms)
   PASSED: test_toString (16ms)
   PASSED: test_toString_2 (0ms)
   PASSED: test_toString_3 (0ms)
   PASSED: test_toString_4 (0ms)
   PASSED: test_toString_5 (0ms)
   PASSED: test_toString_6 (0ms)
   PASSED: test_toString_7 (0ms)
   PASSED: test_toString_8 (0ms)
=  MatrixTest: 19/20 test cases passed (16ms)
```

1.6 How to validate common test units specifically

A test provides various methods to test the test units.

Example (boolean)

```
var boolean1 = true;
var boolean2 = false;

//Expects that ‘bool1’ is true.
expect(boolean1);

//Expects that ‘bool2’ is false.
expectNot(boolean2);
```

Example (int)

```

//Expects that an int is positive.
expect(15).isPositive();

//Expects that an int is bigger than 10.
expect(15).isBiggerThan(10);
```

Example (string)

```

//Expects that a string is not empty.
expect(“Hello World!”).isNotEmpty();

//Expects that a string is not longer than 20 characters.
expect(“Hello World!”).isNotLongerThan(20);
```

1.7 How to test floating point numbers with deviations

Floating point numbers can have rounding errors, especially when they are calculated by a complex algorithm. For example, when the ideal result is 2.0, but 1.999999 or 2.000001 comes out.

Tests provide functionalities to test floating point numbers with deviations. A value is regarded as correct, when it does not deviate more than the default deviation or a given maximum deviation. The default deviation is 10^{-9}.

Example

```
var result = getSquareRoot(2.0);

//Expects that the result is 1.4142 with a maximum deviation of 0.001.
expect(result).withMaxDeviation(0.001).isEqualTo(1.4142);
```

1.8 How to test methods on exceptions

Tests are able to test whether a method will throw exceptions.

Example

```
//Expects that dividing by 0 throws an arithmetic exception.
expect(() -> 10 / 0).throwsExceptionOfType(ArithemeticException.class);

//Expects that dividing by 2 throws no exception.
expect(() -> 10 / 2).doesNotThrowException();
```

2 Test pools

2.1 What, why and how

What a test pool is

A test pool can contain test classes and can run their tests.

Why to use test pools

Test pools can bundle test classes whose all tests should be run. Test pools provide a common and comfortable way to bundle test classes.

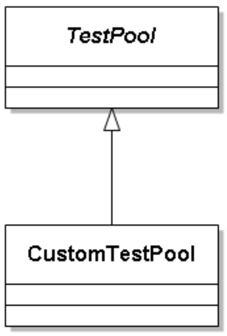
How to use test pools

To use test pools, import the Nolix library into your project. Then you can access the test pool in your project. The test pool can be found in the package ‘ch.nolix.primitive.testoid’.

2.2 How to define a custom test pool

A custom test pool can be defined by inheriting from the test pool class. A test pool is empty at the beginning.

Example



```
import ch.nolix.core.test.TestPool;

public class CustomTestPool extends TestPool {

}
```

2.3 How to add tests to a test pool

A test pool can contain several tests. Tests cannot be added to a test pool from outside. So the tests of a test pool must be added in the constructor.

Example

Lets add 2 tests to our custom test pool.

```
import ch.nolix.core.test.TestPool;

public class CustomTestPool extends TestPool {

	//constructor
	public CustomTestPool() {
		addTestClass(
			CustomTest.class,
			MatrixTest.class
		);
	}
}
```

The method ‘addTest’ can take an arbitrary number of tests.

2.4 How to create and run a test pool

To run the tests of a test pool, an instance of the test pool must be created and its run method must be called.

Example

```
var testPool = new CustomTestPool();
testPool.run();
```

This can be done in 1 line.

```
new CustomTestPool().run();
```

The tests of the test pool are run the same way as when they were created and run separately one after one. The tests of a test pool are run in the same order how they were added to the test pool.

2.5 How to nest test pools

A test pool can also contain other test pools. Test pools cannot be added to another test pool from outside. The inner test pools a test pool contains must be added in the constructor. A test pool can contain both, tests and other test pools. The method 'addTestPool' can add an arbitrary number of test pools. A test pool cannot contain itself recursively to avoid circular run calls. If there is tried to add a test pool that violates this condition, the method 'addTestPool' throws an exception.

Example

Lets add two other test pools to our custom test pool.

```
import ch.nolix.core.test.TestPool;

public class CustomTestPool extends TestPool {

     //constructor
     public CustomTestPool() {

           addTestPool(
                 new CustomSubTestPool1(),
                 new CustomSubTestPool2()
           );

           addTestClass(
                 CustomTest.class,
                 MatrixTest.class
           );
     }
}
```