JUnit

why do we need a testing framework?

- easy
- fast
- often
- fast feedback

what is included?

- assertions
- runners
- rules
- suits (test aggregators)

```java
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CalculatorTest {

  private Calculator calculator;

  @Before
  public void createCalculator() {
    calculator = new Calculator();
  }

  @After
  public void removeCalculator() {
    calculator.cleanupResources();
  }

  @Test
  @Ignore
  public void shouldAddTwoNumbers() {
    Integer result = calculator.add(2, 3);

    assertEquals(new Integer(5), result);
  }

  @Test(expected = IllegalArgumentException.class)
  public void shouldNotDivadeByZero() {
    Integer result = calculator.div(2, 0);
  }
}
```

```
public class CalculatorTest {

  private Calculator calculator;

  @Before
  public void createCalculator() {
    calculator = new Calculator();
  }

  ...

}
```

Method executed before each test

```
public class CalculatorTest {

    private Calculator calculator = new Calculator();
    ...

}
```

Field initialised before each test

```
public class CalculatorTest {

  private Calculator calculator;

  @After
  public void removeCalculator() {
    calculator.cleanupResources;
  }

  ...

}
```

Method executed after each test

```java
public class CalculatorTest {

  private Calculator calculator = new Calculator();

  @Test
  @Ignore
  public void shouldAddTwoNumbers() {
    Integer result = calculator.add(2, 3);

    assertEquals(new Integer(5), result);
  }

  @Test(expected = IllegalArgumentException.class)
  public void shouldNotDivideByZero() {
    Integer result = calculator.div(2, 0);
  }

}
```

Test method:

- annotated @Test
- parameterless
- public
- void
- any name
- any exception

```
public class CalculatorTest {

  @Test
  @Ignore
  public void shouldAddTwoNumbers() {
    Integer result = calculator.add(2, 3);

    assertEquals(new Integer(5), result);
  }

  @Test(expected = IllegalArgumentException.class)
  public void shouldNotDivideByZero() {
    Integer result = calculator.div(2, 0);
  }

}
```

Ignored - won't be executed

Will succeed if the defined exception is thrown

additional methods

`@BeforeClass` `public static void name()`	Executed once, before all test methods
`@AfterClass` `public static void name()`	Executed once, after all test methods

Traditional assertions

- `assertTrue (actualValue)`
- `assertFalse (actualValue)`
- `assertEquals (expectedValue, actualValue)`
- `assert(Not)Same (expectedObject, actualObject)`
- `assert(Not)Null (actualObject)`
- `assertArrayEquals (expectedArray, actualArray)`

All assertions have an optional parameter - message,
which is shown to the user in case test fails.

Excercise

traditional assertions

```
public static String revert(String word)
public static boolean revertable(String word)
```

```
@Test
public void revertedNullIsNull()

@Test
public void emptyWordIsNotRevertable()

@Test
public void notEmptyWordIsRevertable()

@Test
public void revertedPalindromeIsEqualButNotSame()
```

assertThat - Hamcrest

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import static org.junit.matchers.JUnitMatchers.*;
...
assertThat(actual, is(expected));
assertThat(actual, sameInstance(expected));
assertThat(actual, not(expected));
assertThat(actual, instanceOf(String.class));
assertThat(actual, nullValue());
assertThat(actual, notNullValue());
assertThat(strings, hasItems("1"));
assertThat(actual, containsString("substring"));
assertThat(actual, either(nullValue()).or(is("actual")));
assertThat(strings, both(instanceOf(List.class)).and(hasItems("1")));
```

- hamcrest included from version 4.4
- assertions based on passing Matchers
- more DSLy
- expected and actual parameters have switched positions

context assertions from fest-assert

```
import static org.fest.assertions.Assertions.*;
...
assertThat(object);
```

- generic method which returns a specific assertion object, based on the parameter type
- apart from primitives, the assertions are available for:

  - Object
  - String
  - Throwable
  - BigDecimal
  - File
  - [ ]
  - Collection
  - List
  - Map
  - Image

fest-assert for a list

```
assertThat(new ArrayList<String>())
  .startsWith("a")
  .endsWith("c")
  .contains("a","b")
  .containsOnly("a")
  .containsExactly("a","b")
  .containsSequence("a","b")
  .isNotNull()
  .isNotSameAs(new ArrayList<String>())
  .isNotEmpty();

assertThat(someObject).isNull();
```

fest-assert onProperty asserting by some property value

```
assertThat(new ArrayList<Person>())
  .onProperty("name")
  .contains("Наталия","Наташа","Наталочка")
```

PRAGMATISTS

fest-assert with conditions

- we call a method on the assertion object
  - is (condition) / isNot (condition)
  - satisfies (condition) / doesNotSatisfy(condition)
- The parameter is the condition object:

```
assertThat(list).is(containingOnlyStrings());

private Condition<List> containingOnlyStrings() {
  return new Condition<List>() {
    public boolean matches(List values) {
      for (Object value : values) {
        if (value instanceof String == false)
          return false;
      }
      return true;
    }
  }
}
```

JUnit - assumptions

Initial conditions for a test
If not satisfied, the test is not executed

```
import static org.junit.Assume.*;

@Test
public void filenameIncludesUsernameOnUnix() {
    assumeThat(File.separatorChar, is('/'));

    assertThat(getConfigFileName(), is("configfiles/config.xml"));
}

@Test
public void filenameIncludesUsernameOnWindows() {
    assumeThat(File.separatorChar, is('\\'));

    assertThat(getConfigFileName(), is("configfiles\\config.xml"));
}
```

Parameterised tests

```
public class Person {

    public Person(int age) {
        // TODO Auto-generated method stub
    }

    public boolean isAdult() {
        // TODO Auto-generated method stub
        return false;
    }
}
```

Person class:

* the person has some age

* if the age >= 18, the person
  is an adult

Standard parameterized tests

```
@RunWith(Parameterized.class)
public class PersonJUnitParametrizedTest {

    private final boolean isAdult;
    private final Person person;

    public PersonJUnitParametrizedTest(Person person, boolean isAdult) {
        this.person = person;
        this.isAdult = isAdult;
    }

    @Test
    public void shouldCheckIfIsAdult() {
        assertThat(person.isAdult(), is(isAdult));
    }

    @Parameters
    public static List<Object[]> params() {
        return Lists.newArrayList($(new Person(12), false),
                $(new Person(34), true));
    }
}
```

Definition of a Runner

Standard parameterized tests

```
@RunWith(Parameterized.class)
public class PersonJUnitParametrizedTest {

    private final boolean isAdult;
    private final Person person;

    public PersonJUnitParametrizedTest(Person person, boolean isAdult) {
        this.person = person;
        this.isAdult = isAdult;
    }

    @Test
    public void shouldCheckIfIsAdult() {
        assertThat(person.isAdult(), is(isAdult));
    }

    @Parameters
    public static List<Object[]> params() {
        return Lists.newArrayList($(new Person(12), false),
                $(new Person(34), true));
    }
}
```

Definition of test parameters

Standard parameterized tests

```java
@RunWith(Parameterized.class)
public class PersonJUnitParametrizedTest {

    private final boolean isAdult;
    private final Person person;

    public PersonJUnitParametrizedTest(Person person, boolean isAdult) {
        this.person = person;
        this.isAdult = isAdult;
    }

    @Test
    public void shouldCheckIfIsAdult() {
        assertThat(person.isAdult(), is(isAdult));
    }

    @Parameters
    public static List<Object[]> params() {
        return Lists.newArrayList($(new Person(12), false),
               $(new Person(34), true));
    }
}
```

A method that returns a list of parameter sets

Requirements:
- @Parameters
- static
- public
- returns List<Object[]>

Standard parameterized tests

```
@RunWith(Parameterized.class)
public class PersonJUnitParametrizedTest {

    private final boolean isAdult;
    private final Person person;

    public PersonJUnitParametrizedTest(Person person, boolean isAdult) {
        this.person = person;
        this.isAdult = isAdult;
    }

    @Test
    public void shouldCheckIfIsAdult() {
        assertThat(person.isAdult(), is(isAdult));
    }

    @Parameters
    public static List<Object[]> params() {
        return Lists.newArrayList($(new Person(12), false),
                $(new Person(34), true));
    }
}
```

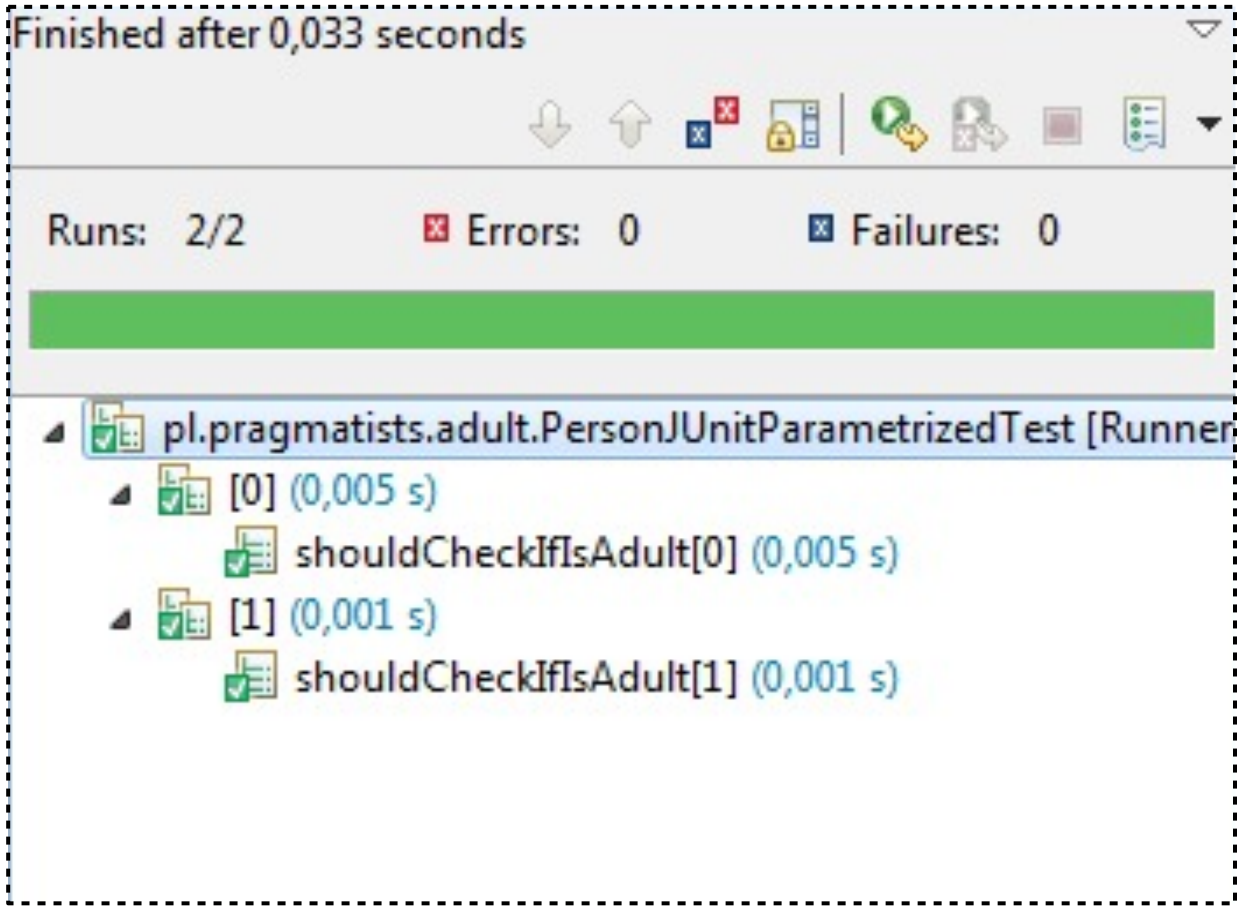
The constructor has parameters, matching the @Parameters method return values.

Standard parameterized tests

```
@RunWith(Parameterized.class)
public class PersonJUnitParametrizedTest {

    private final boolean isAdult;
    private final Person person;

    public PersonJUnitParametrizedTest(Person person, boolean isAdult) {
        this.person = person;
        this.isAdult = isAdult;
    }

    @Test
    public void shouldCheckIfIsAdult() {
        assertThat(person.isAdult(), is(isAdult));
    }

    @Parameters
    public static List<Object[]> params() {
        return Lists.newArrayList($(new Person(12), false),
                $(new Person(34), true));
    }
}
```

In the end the test method is executed. It uses the object fields, which are the parameters.

Standard parameterized tests



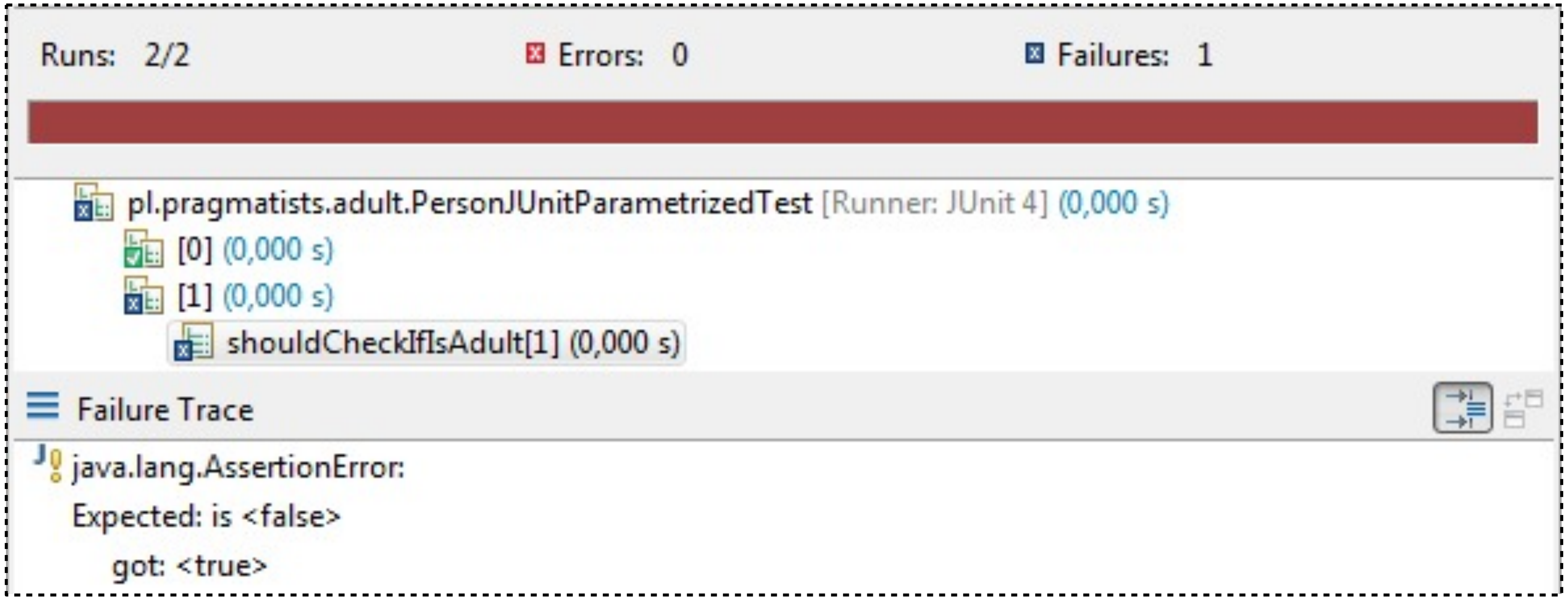
```
Finished after 0,033 seconds

Runs:   2/2          Errors:   0          Failures:   0

pl.pragmatists.adult.PersonJUnitParametrizedTest [Runner
  [0] (0,005 s)
    shouldCheckIfIsAdult[0] (0,005 s)
  [1] (0,001 s)
    shouldCheckIfIsAdult[1] (0,001 s)
```

JUnit shows the results for each test run

PRAGMATISTS

www.pragmatists.pl

Standard parameterized tests



```
Runs: 2/2        Errors: 0        Failures: 1

pl.pragmatists.adult.PersonJUnitParametrizedTest [Runner: JUnit 4] (0,000 s)
    [0] (0,000 s)
    [1] (0,000 s)
        shouldCheckIfIsAdult[1] (0,000 s)
```

Failure Trace

```
java.lang.AssertionError:
Expected: is <false>
     got: <true>
```

We learn for which parameter set the test fails.

Theories

```
@RunWith(Theories.class)
public class PersonJUnitTheoriesTest {
    @DataPoints
    public static Object[][] people = new Object[][] {
            $(new Person(12), false), $(new Person(34), true) };

    @Theory
    public void shouldCheckIfIsAdult(Object[] params) {
        assertThat(((Person) params[0]).isAdult(), is((Boolean) params[1]));
    }
}
```

Runner definition

Theories

Definition of test data

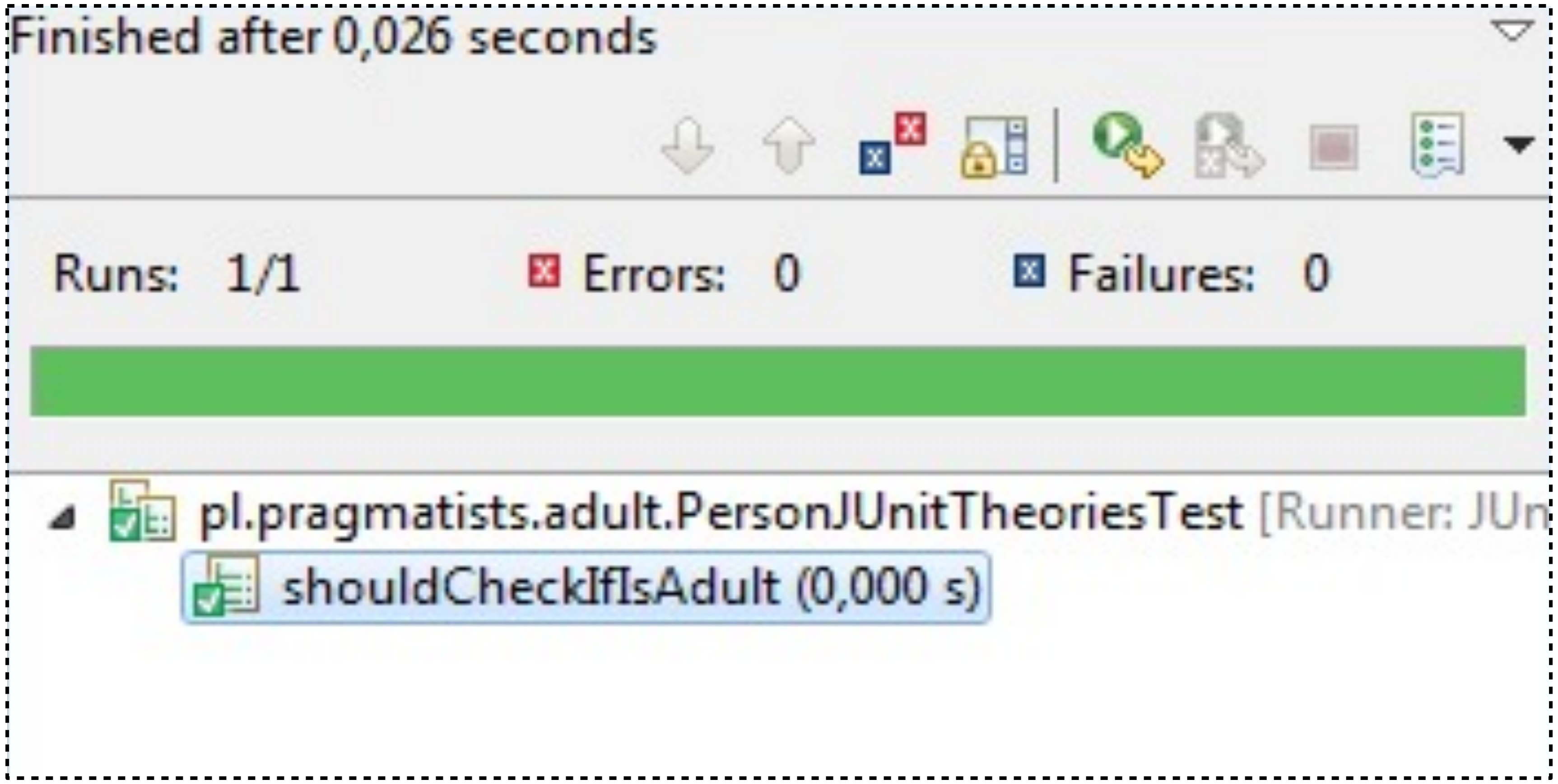
```
@RunWith(Theories.class)
public class PersonJUnitTheoriesTest {
    @DataPoints
    public static Object[][] people = new Object[][] {
            $(new Person(12), false), $(new Person(34), true) };

    @Theory
    public void shouldCheckIfIsAdult(Object[] params) {
        assertThat(((Person) params[0]).isAdult(), is((Boolean) params[1]));
    }
}
```

Theories

```
@RunWith(Theories.class)
public class PersonJUnitTheoriesTest {
    @DataPoints
    public static Object[][] people = new Object[][] {
            $(new Person(12), false), $(new Person(34), true) };

    @Theory
    public void shouldCheckIfIsAdult(Object[] params) {
        assertThat(((Person) params[0]).isAdult(), is((Boolean) params[1]));
    }
}
```

@Theory
parameters must have
same types as test data

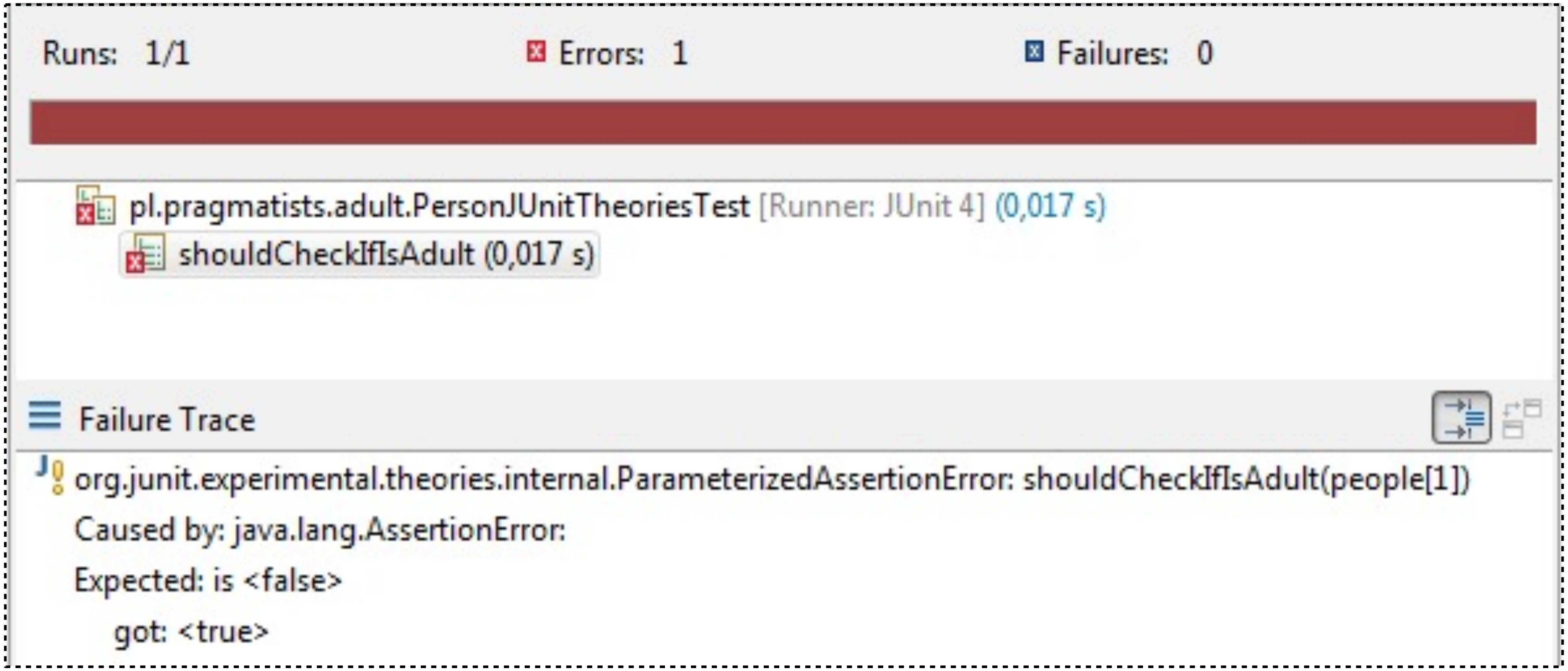
Theories



```
Finished after 0,026 seconds

Runs:   1/1        Errors:   0        Failures:   0

pl.pragmatists.adult.PersonJUnitTheoriesTest [Runner: JUn
    shouldCheckIfIsAdult (0,000 s)
```

JUnit shows only one execution for all parameter sets.

Theories


```
Runs: 1/1        Errors: 1        Failures: 0

pl.pragmatists.adult.PersonJUnitTheoriesTest [Runner: JUnit 4] (0,017 s)
  shouldCheckIfIsAdult (0,017 s)

Failure Trace
org.junit.experimental.theories.internal.ParameterizedAssertionError: shouldCheckIfIsAdult(people[1])
Caused by: java.lang.AssertionError:
Expected: is <false>
     got: <true>
```

If the test fails, we learn from the exception which parameter set failed.

JUnitParams

```
@RunWith(JUnitParamsRunner.class)
public class PersonJUnitParamsTest {
    @Parameters
    @Test
    public void shouldCheckIfIsAdult(Person person, boolean isAdult) {
        assertThat(person.isAdult(), is(isAdult));
    }

    public Object[] parametersForShouldCheckIfIsAdult() {
        return $($(new Person(12), false), $(new Person(34), true));
    }
}
```

Runner definition

JUnitParams

The test

```
@RunWith(JUnitParamsRunner.class)
public class PersonJUnitParamsTest {
    @Parameters
    @Test
    public void shouldCheckIfIsAdult(Person person, boolean isAdult) {
        assertThat(person.isAdult(), is(isAdult));
    }

    public Object[] parametersForShouldCheckIfIsAdult() {
        return $($(new Person(12), false), $(new Person(34), true));
    }
}
```

JUnitParams

```
@RunWith(JUnitParamsRunner.class)
public class PersonJUnitParamsTest {
    @Parameters
    @Test
    public void shouldCheckIfIsAdult(Person person, boolean isAdult) {
        assertThat(person.isAdult(), is(isAdult));
    }

    public Object[] parametersForShouldCheckIfIsAdult() {
        return $($(new Person(12), false), $(new Person(34), true));
    }
}
```

Definition of parameters. The method should return an array of objects. Should be named same as test, but prefixed with „parametersFor".

JUnitParams

You can also specify
the method that provides
the test parameters.

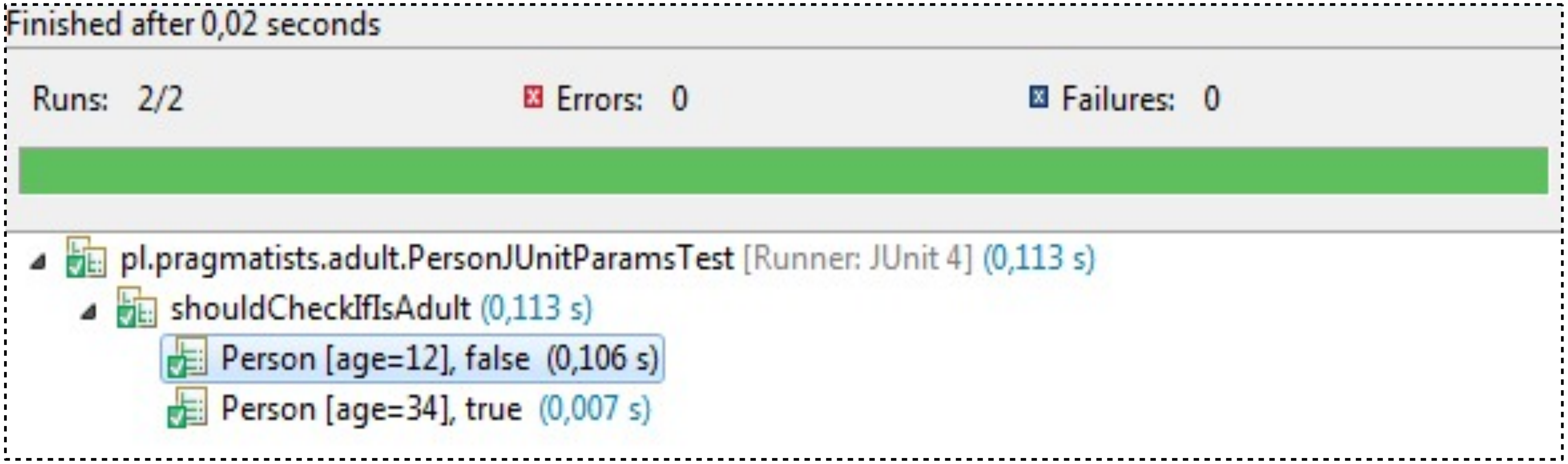
```
@RunWith(JUnitParamsRunner.class)
public class PersonJUnitParamsTest2 {
    @Parameters(method = "people")
    @Test
    public void shouldCheckIfIsAdult(Person person, boolean isAdult) {
        assertThat(person.isAdult(), is(isAdult));
    }

    public Object[] people() {
        return $($(new Person(12), false),
                $(new Person(34), true));
    }
}
```

JUnitParams

You can also specify params directly in the @Parameters annotation.

```
@RunWith(JUnitParamsRunner.class)
public class PersonJUnitParamsTest3 {
    @Parameters({ "12, false", "34, true" })
    @Test
    public void shouldCheckIfIsAdult(int age, boolean isAdult) {
        assertThat(new Person(age).isAdult(), is(isAdult));
    }
}
```

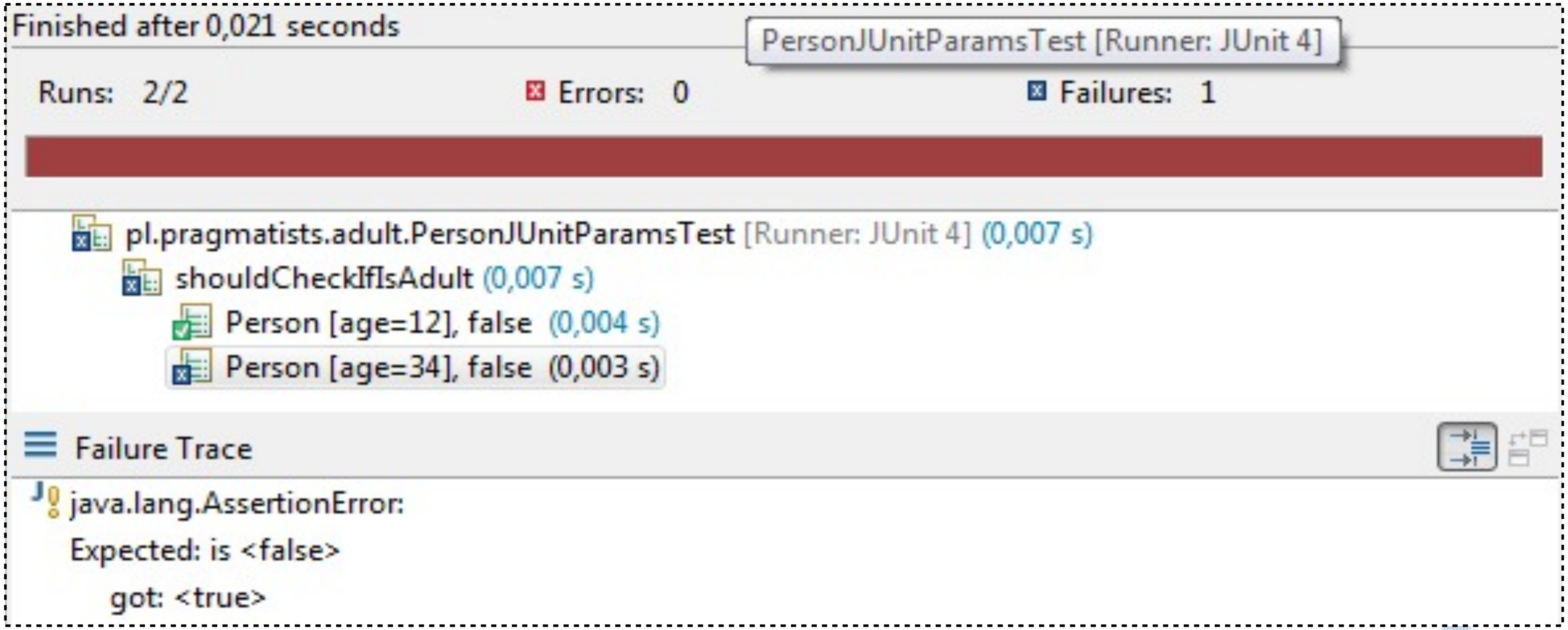
JUnitParams



```
Finished after 0,02 seconds

Runs:   2/2          Errors:   0          Failures:   0

◢ pl.pragmatists.adult.PersonJUnitParamsTest [Runner: JUnit 4] (0,113 s)
   ◢ shouldCheckIfIsAdult (0,113 s)
        Person [age=12], false  (0,106 s)
        Person [age=34], true  (0,007 s)
```

Each test shows results for each parameter set. Additionally the params are printed out, which improves readability.

JUnitParams



```
Finished after 0,021 seconds                PersonJUnitParamsTest [Runner: JUnit 4]

Runs:  2/2            Errors:  0            Failures:  1

pl.pragmatists.adult.PersonJUnitParamsTest [Runner: JUnit 4] (0,007 s)
    shouldCheckIfIsAdult (0,007 s)
        Person [age=12], false  (0,004 s)
        Person [age=34], false  (0,003 s)

Failure Trace
java.lang.AssertionError:
Expected: is <false>
     got: <true>
```

ExpectedException

Rule definition

```
public class MathTest {

    @Rule
    public ExpectedException expectedException = ExpectedException.none();

    @Test
    public void shouldNotAllowDivisionByZero() throws Exception {
        expectedException.expect(ArithmeticException.class);
        expectedException.expectMessage("/ by zero");
        MathUtils.divide(1).by(0);
    }
}
```

Grouping tests

- Suite

```
@RunWith(Suite.class)
@SuiteClasses( { NodeTest.class, LinkTest.class})
public class AllGraphTests {

}
```

- ClasspathSuite

```
import org.junit.extensions.cpsuite.ClasspathSuite.*;
...
@ClassnameFilters({"pragmatists.*", "!.*SlowTest"})
public class FastTestSuite {

}
```