xUnit

Sergey Mechtaev

mechtaev@pku.edu.cn

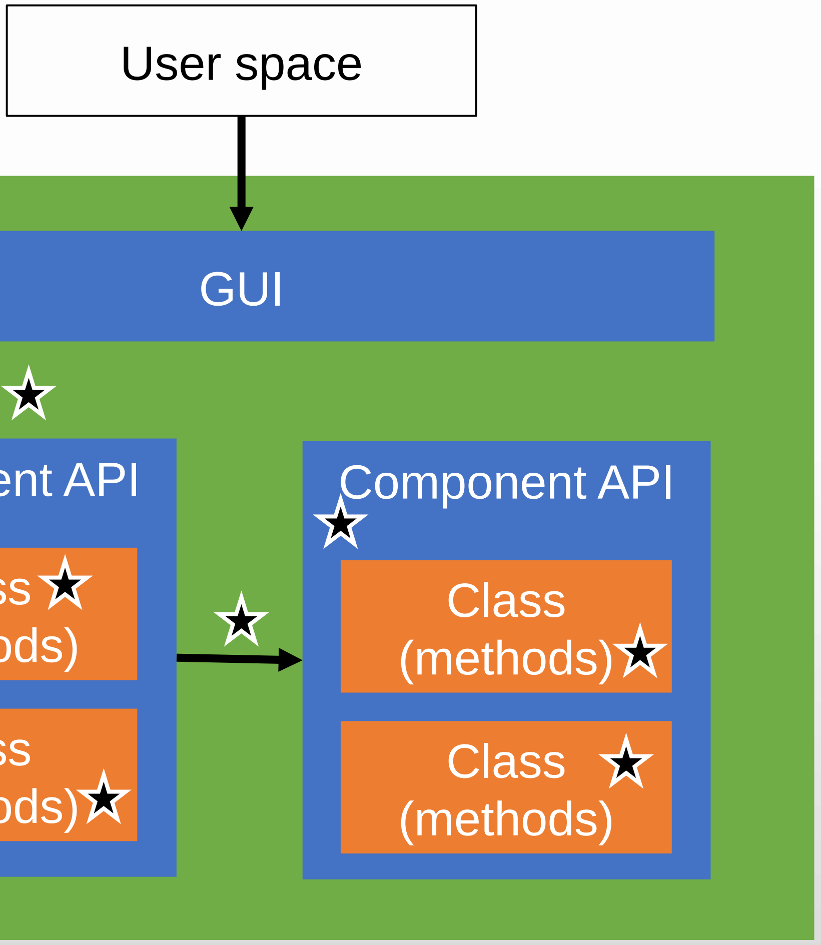
Peking University

xUnit Testing Frameworks

- **SUnit** for Smalltalk created by Kent Beck in 1989.
- **JUnit**, a port for Java, made by Kent Beck and Erich Gamma in 1997.
- Python’s built-in **unittest**, **Pytest**, Rust’s built-in testing framework, etc.

Collectively (sometimes) known as xUnit frameworks.

Where Do We Use JUnit?



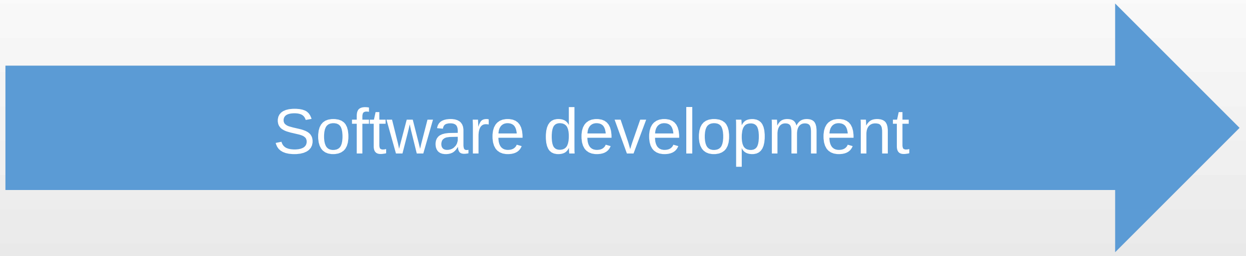
★ Class, interface, method level testing

When Do We Use JUnit?

Test-driven development

Continuous testing

Regression testing



Before

During

After

JUnit vs Println

```
@Test
public void testAdd() {
    Calculator calculator = new Calculator();
    double result = calculator.add(10, 50);
    assertEquals(60, result, 0)
}
```

```
public class CalculatorTest {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        double result = calculator.add(10,50);
        if (result != 60)
            System.out.println("Bad result: " + result);
    }
}
```

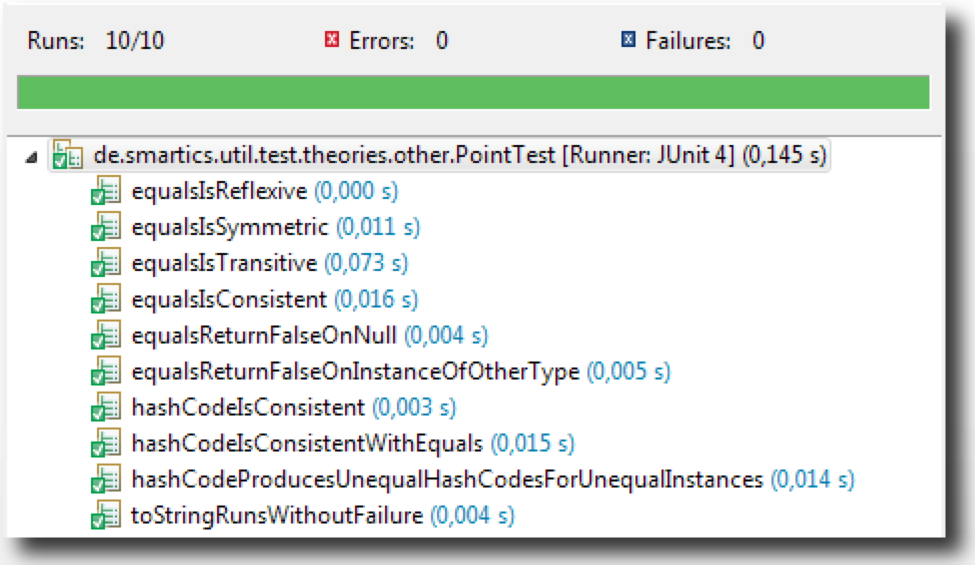
JUnit as Specification

```
/**
* Purpose: program for wrapping strings on spaces an
d
indenting strings if we
* break them before '+' or '=' symbols as in Java.
*/
public class Wrapper {
    public String wrap(String s, int length) {
    ...
    }
}
```

JUnit as Specification

```
@Test
public void testWrapNull() {
    assertEquals("", wrapper.wrap(null, 10));
}
@Test
public void testOverTheLimitShouldWrapAtSecondWord() {
    assertEquals("word word\nword",
                 wrapper.wrap("word word word", 5));
}
@Test
public void testLongerThanLimitShouldNotWrap() {
    assertEquals("word word", wrapper.wrap("word word", 6));
}
```

Results and Statistics


```
Runs:  10/10          Errors:  0          Failures:  0

de.smartics.util.test.theories.other.PointTest [Runner: JUnit 4] (0,145 s)
    equalsIsReflexive (0,000 s)
    equalsIsSymmetric (0,011 s)
    equalsIsTransitive (0,073 s)
    equalsIsConsistent (0,016 s)
    equalsReturnFalseOnNull (0,004 s)
    equalsReturnFalseOnInstanceOfOtherType (0,005 s)
    hashCodeIsConsistent (0,003 s)
    hashCodeIsConsistentWithEquals (0,015 s)
    hashCodeProducesUnequalHashCodesForUnequalInstances (0,014 s)
    toStringRunsWithoutFailure (0,004 s)
```

- Execution time
- Failures/Errors
- Stack traces
- Contrast expected and actual

Test Method

```
    @Test
    public void testAdd() {
1.      Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
2.      assertEquals(60, result, 0)
3.  }
```

1. Object under test
2. Method under test with parameters
3. Comparison of expected and actual data

Assert

```
assertEquals("", result);
```

“check if **expected** and **actual** values are **equal”**

Assert Methods

Equality for object, int, long, and byte values, array:

```
assertEquals("strings are not equal", "text", "text")
;
```

For Boolean values:

```
assertTrue("should be true", true);
assertFalse("should be false", false);
```

For object references:

```
assertNull("should be null", null);
assertNotNull("should not be null", new Object());
assertNotSame("should not be same Object",
new Object(), new Object());
Integer aNumber = Integer.valueOf(768);
assertSame("should be same", aNumber, aNumber);
```

Will The Test Fail?

```
public void testEquality() {
    String a = "abcde";
    String b = new String(a);
    assertTrue(a.equals(b));
    assertFalse(a == b);
    assertEquals(a, b);
    String c = "abcde";
    assertNotSame(a, b);
    assertSame(a,c);
}
```

Test Class

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {
    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals(60, result, 0);
    }
}
```

Setup and Teardown

```
public class Example {
    File output;
    @Before
    public void createOutputFile() {
        output= new File(...);
    }
    @Test
    public void something() {
        ...
    }
    @After
    public void deleteOutputFile() {
        output.delete();
    }
}
```

Setup executed before each test

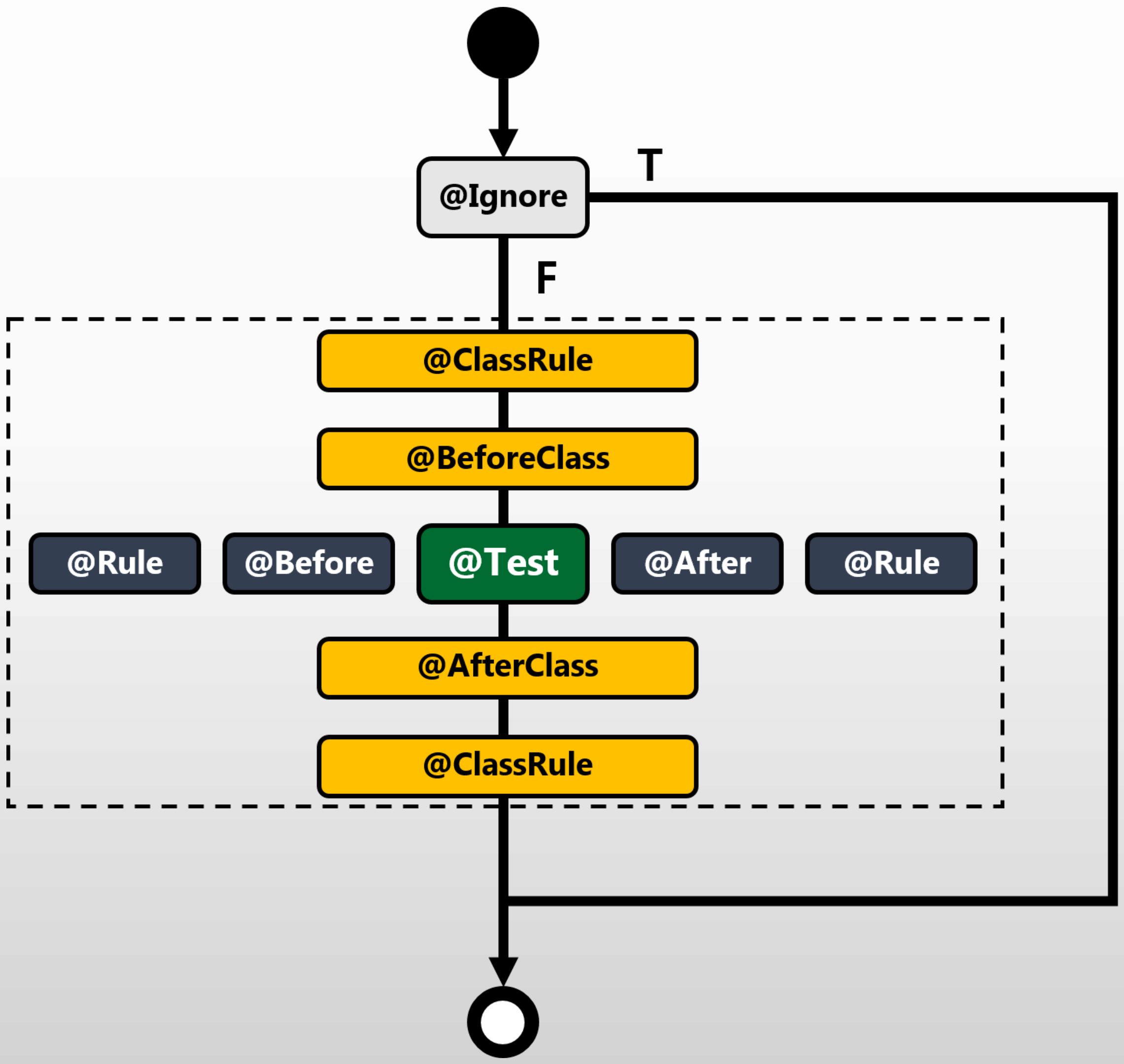
Teardown executed after each test

Rules

```
public static class HasTempFolder {
    @Rule
    public final TemporaryFolder folder =
        new TemporaryFolder();

    @Test
    public void testUsingTempFolder() throws IOExcepti
on {
      File createdFile = folder.newFile("myfile.txt");
      File createdFolder = folder.newFolder("subfolder
");
      // ...
    }
}
```

Execution Model



Test Suite

Allows grouping test classes in different sets for test execution:

```
@RunWith(Suite.class)
@SuiteClasses({CSVRendererTest.class,EmacsRenderer
erTest.class,XMLRendererTest.class,TextPadRender
erTest.class})
public class RenderersTests {
}
```

Exception Testing

```
@Test(expected = IndexOutOfBoundsException.class)
public void elementAt() {
    int i = fFull.get(0);
    assertTrue(i == 1);
    fFull.get(fFull.size());
}
```

Method fail()

```
@Test
public void testBogusArguments() {
    try {
        tokenFilterFactory("Normalization",
                           "bogusArg", "bogusValue");
        fail();
    } catch (IllegalArgumentException expected) {
        assertTrue(expected.getMessage().contains(
                        "Unknown parameters"));
    }
}
```

Timeout Testing

```
@Test(timeout=1000)
public void testComputationWithSharing()
               throws Exception {
    int size = 30;
    Expression expr = buildExpressionWithSharing(size);
    assertEquals(IntConst.of(1<<size), expr.compute());
}
```

Python’s unittest

```
import unittest

class DefaultWidgetSizeTestCase(unittest.TestCase):

    def test_default_widget_size(self):
        widget = Widget('The widget')
        r = widget.size()
        self.assertEqual(r, (50, 50))
```

Test class is inherited from unittest.TestClass

Test method's name starts with "test_"

Object under test

Method under test

Comparison of the results

Assert methods

Method	Checks that
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	not isinstance(a, b)

SetUp

```
import unittest

class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')
    def test_default_widget_size(self):
        self.assertEqual(self.widget.size(), (50,50))
    def test_widget_resize(self):
        self.widget.resize(100,150)
        self.assertEqual(self.widget.size(), (100,150))
```

Executed before each test

TearDown

```
import unittest

class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget('The widget')
    def tearDown(self):
        self.widget.dispose()
```

$\xleftarrow{\hspace{3cm}}$ Executed after each test

Testing Exceptions

```
import unittest

class TestStringMethods(unittest.TestCase):

   def test_split(self):
      s = 'hello world'
      self.assertEqual(s.split(), ['hello', 'world'])
      # check that s.split fails when the separator is not a string
      with self.assertRaises(TypeError):
         s.split(2)
```

Testing exception with a context manager

Best Practices

- One **@Test** - One [feature/class/object] under test
- Strive to write short test cases (~ **5LOC** long)
- Visualize data in test cases
- Choose meaningful test method names
- Don't repeat yourself
    - Put common parts in setup and teardown
- Put test cases in the same package structure as source code. Test code is separate, but you can access methods with package accessibility

Visualizing Data

```java
public class TestGJChronology extends TestCase {

    private static final DateTimeZone PARIS = DateTimeZone.forID("Europe/Paris");
    private static final DateTimeZone LONDON = DateTimeZone.forID("Europe/London");
    private static final DateTimeZone TOKYO = DateTimeZone.forID("Asia/Tokyo");

    long y2002days = 365 + 365 + 366 + 365 + 365 + 365 + 366 + 365 + 365 + 365 +
                     366 + 365 + 365 + 365 + 366 + 365 + 365 + 365 + 366 + 365 +
                     365 + 365 + 366 + 365 + 365 + 365 + 366 + 365 + 365 + 365 +
                     366 + 365;
    // 2002-06-09
    private long TEST_TIME_NOW =
            (y2002days + 31L + 28L + 31L + 30L + 31L + 9L -1L) *
              DateTimeConstants.MILLIS_PER_DAY;

    private DateTimeZone originalDateTimeZone = null;

    ...
```