Junit

14.-15.4.2016

Ján Švantner
CIIT

Unit Testing in General

- Motivations
  - Whitebox testing
  - Divide & Conquer
- Problems
  - Testing of DB
  - Testing of Frontend
  - Mocking of the return values can be complex
- Notes
  - Independence of tests
  - Test Pyramid
  - Execution time
  - Test coverage
  - Maintainability

Common Annotations

@Test

@Before

@BeforeClass

@After

@AfterClass

@Ignore

@SuiteName

@Test(timeout=500)

- test will fail after timeout

@Test(expected=IllegalArgumentException.class)

- test expects an exception to be thrown

Order

- What is order of execution?

  @Before

  @BeforeClass

  @After

  @AfterClass

- Inheritance?

Test order

- @FixMethodOrder(MethodSorters.JVM):
  - Leaves the test methods in the order returned by the JVM. This order may vary from run to run.
- @FixMethodOrder(MethodSorters.NAME_ASCENDING)

Exception-testing

- @Test (expected= IndexOutOfBoundsException.class)
- Try / catch idiom:

```
try {
   testedMethodWhichShouldThrowAnExcepion();
   fail("Expected exception to be thrown");
} catch (Exception e) {
   assertThat(e.getMessage(),
   is(„message"));
}
```

Parametrized Tests

```
@RunWith(Parameterized.class)
public class FibonacciTest {
   @Parameters
   public static Collection<Object[]> data() {
      return Arrays.asList(new Object[][] { { 0, 0 }, { 1, 1 }, { 2, 1
      }, { 3, 2 }, { 4, 3 }, { 5, 5 },{ 6, 8 } });
   }
   @Parameter(0)
   public /* NOT private */ int fInput;

   @Parameter(value = 1)
   public /* NOT private */ int fExpected;

   @Test
   public void test() {
      assertEquals(fExpected, Fibonacci.compute(fInput));
   }
}
```

Assumptions

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


@Test
public void filenameIncludesUsername() {
   assumeThat(File.separatorChar, is('/'));
   String fn = new
   User("optimus").configFileName();
   assertThat(fn, is("configfiles/optimus.cfg"));
}
```

Rules

- Reusable @Before and @After functionality

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

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

Exception Testing with Rules

```
@Rule
public ExpectedException thrown =
ExpectedException.none();


@Test
public void shouldTestExceptionMessage() throws
IndexOutOfBoundsException {
   List<Object> list = new ArrayList<Object>();
   thrown.expect(IndexOutOfBoundsException.class);
   thrown.expectMessage("Index: 0, Size: 0");
   list.get(0);
   // execution will never get past to this line
}
```

Default Rules

- `ExternalResource`
- `ErrorCollector`
- `Timeout`
- `ExpectedException`
- `RuleChains`
- `LoggingRule`

Exception Testing with Rules

```
@Rule
public ExpectedException thrown =
ExpectedException.none();


@Test
public void shouldTestExceptionMessage() throws
IndexOutOfBoundsException {
   List<Object> list = new ArrayList<Object>();
   thrown.expect(IndexOutOfBoundsException.class);
   thrown.expectMessage("Index: 0, Size: 0");
   list.get(0);
   // execution will never get past to this line
}
```

Custom Rules

```
public class MyRule implements TestRule {
 @Override
 public Statement apply( Statement base, Description description ) {
  return new MyStatement( base );
 }
}

public class MyStatement extends Statement {
 private final Statement base;

 public MyStatement( Statement base ) {
  this.base = base;
 }

 @Override
 public void evaluate() throws Throwable {
  System.out.print( "before" );
  try {
    base.evaluate();
  } finally {
    System.out.print( "after" );
  }
 }
}
```

Custom Rule Usage

```
public class MyTest {
   @Rule
   public MyRule myRule = new MyRule();

   @Test
   public void testRun() {
      System.out.print( " during " );
   }
}
```

- prints : before during after

Theories

```
@Theory
public void multiplyIsInverseOfDivideWithInlineDataPoints(
   @TestedOn(ints = {0, 5, 10}) int amount,
   @TestedOn(ints = {0, 1, 2}) int m )
{
   assumeThat(m, not(0));
   assertThat(new
   Dollar(amount).times(m).divideBy(m).getAmount(),
   is(amount));
}
```

- Between :

```
@Between(first = -100, last = 100) int amount
```

Mocks

- Dividing the code base
- Verification of calls
- Mocking of the private & static methods
- Implementations
  - Mockito
  - EasyMock
  - PowerMock

Builder pattern

- Constructors with many parameters are not readable well

```
new Person(„John“, „Cassidy“, 15, 43, 150, „green“)
```

- Builder allows us to define parameters in more convenient way

```
new PersonBuilder()
withFirstName(„John“).withSurname(„Cassidy“).
withAge(15).withWeight(43).withHeight(150).
withEyeColor(„green“).build();
```

- Allow us to set only parameters important for test:

```
new PersonBuilder().withSurname(„Cassidy“).build();
```

Let’s Go Parallel

- http://tempusfugitlibrary.org/documentation/junit/parallel/

```
@RunWith(ConcurrentTestRunner.class)
public class ErrorFlowServiceTest extends
FlowServiceTestBase {


}
```

Sources

- External sources:

  https://github.com/junit-team/junit/wiki

  http://cwd.dhemery.com/2010/12/junit-rules/

  http://tempusfugitlibrary.org/documentation/junit/parallel/

- Presentation sources:

  https://github.com/jansvantner/course-junit