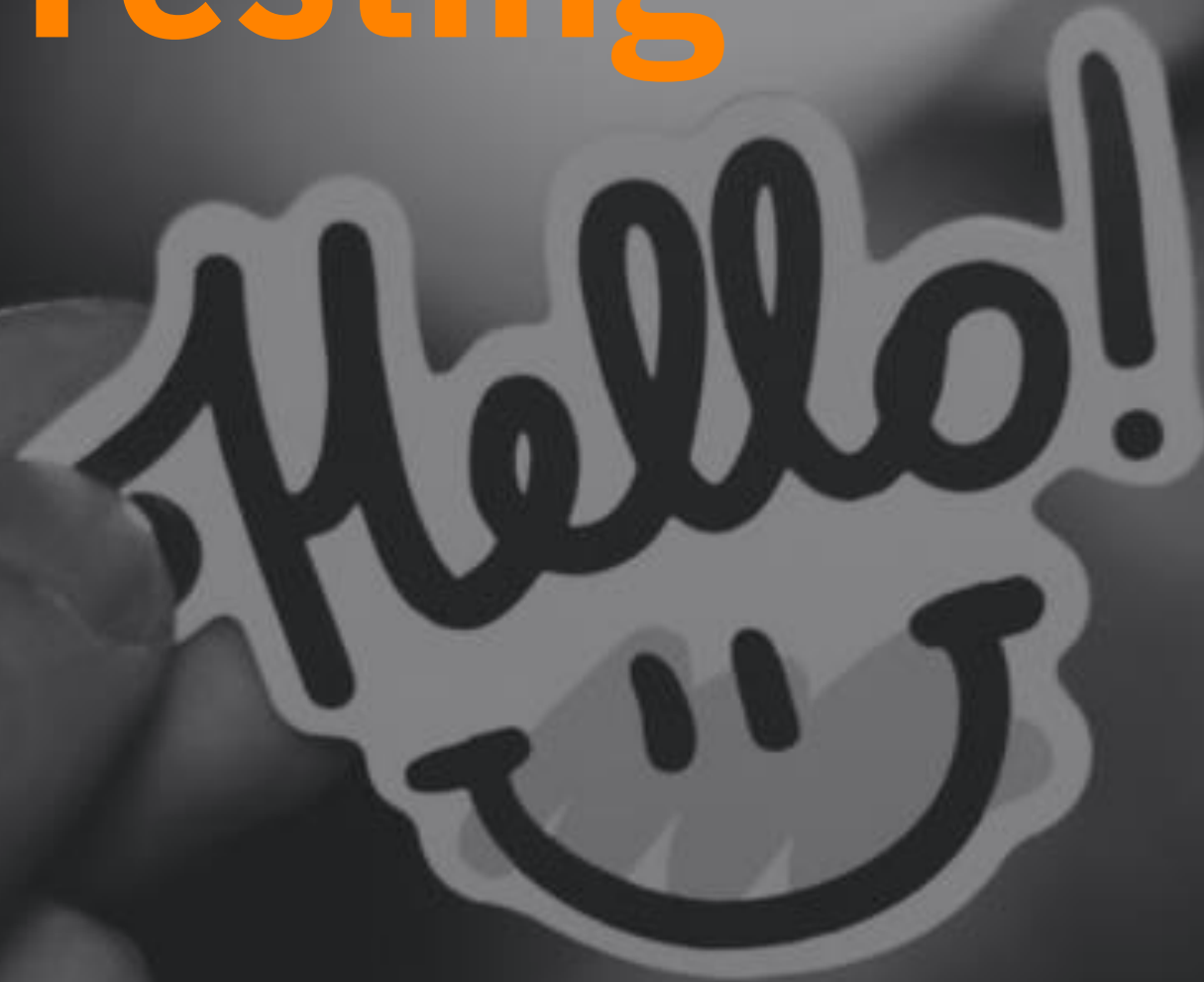


# Next Level Unit Testing



—  
Dominik Poljak, JavaCro, Umag, 14.5.2019.  
dominik.poljak@ecx.io

Who **frequently** writes  
JUnit tests?

# Why Unit Testing?



**The ratio of time spent reading  
(code) versus writing is well  
over 10 to 1...**

– Robert C. Martin, Clean Code



## Benefits of Unit Testing

### Code Quality

- Exposing edge cases

### Finding Bugs

- bugs are found at early stage

### Simplified Integration

- Better interfaces

### Provides Documentation

- Documentation can be generated

### Improves Debugging

- Easier to find issues

**How to get the most  
Value  
with  
Minimal Effort?**



## How to increase value and reduce effort while writing JUnit tests?

### Increase READABILITY

- Introduce common structure

### Make them EASY to write

- Use powerful and simple libraries

### Make logic SIMPLE

- Make them independent (use mocking)

### REUSE them as documentation

- Automate everything you can



## Agenda

# **JUnit 4 Test Structure**

# **Test Simplification via Mocking**

# **Documentation Generation**





# **JUnit4 Test Structure**

## JUnit Test Setup

### Goal:

**Reusable**  
**Configurable**  
**Fast**

### How:

**Common Data**  
**Common Objects**  
**(Mocks e.g. AemMocks)**  
**Utilities**

```
{
  "items": [
    {
      "title": "Lorem ipsum dolor sit amet.",
      "imgSrc": "static/images/media/dummy_produk_300x208.png",
      "campaign": false,
      "budget_price": true,
      "reduced_price": true,
      "campaign_price": false,
      "lactose_free": false,
      "rollover": {
        "rating": {
          "stars": 4,
          "count": 4.4
        },
        "headline": "Creme",
        "price": "2,90 €",
        "list": [
          "Lorem ipsum duis leo",
          "Lorem ipsum sedisfition fritata"
        ]
      }
    },
    {
```





What is **BDD**?



## Behavioral Driven Development (BDD)

- evolved from **TDD**
- **shared language** between tech and non-tech teams
- **behavior** focused

## Simple BDD Test Structure

```
@Test
public void test_when_resource_is_not_valid() throws ServletException, IOException {
    //given
    initInvalidResource();
    initRequest();
    mockImageScaleUtils();

    //when
    servlet.doGet(context.request(), context.response());

    //then
    assertEquals(SlingHttpServletResponse.SC_BAD_REQUEST, context.response().getStatus());
}
```

# AssertJ

## Fluent assertions for java

```
// entry point for all assertThat methods and utility methods (e.g. entry)
import static org.assertj.core.api.Assertions.*;

// basic assertions
assertThat(frodo.getName()).isEqualTo("Frodo");
assertThat(frodo).isNotEqualTo(sauron);

// chaining string specific assertions
assertThat(frodo.getName()).startsWith("Fro")
    .endsWith("do")
    .isEqualToIgnoringCase("frodo");

// collection specific assertions (there are plenty more)
// in the examples below fellowshipOfTheRing is a List<TolkienCharacter>
assertThat(fellowshipOfTheRing).hasSize(9)
    .contains(frodo, sam)
    .doesNotContain(sauron);
}
```

# AssertJ

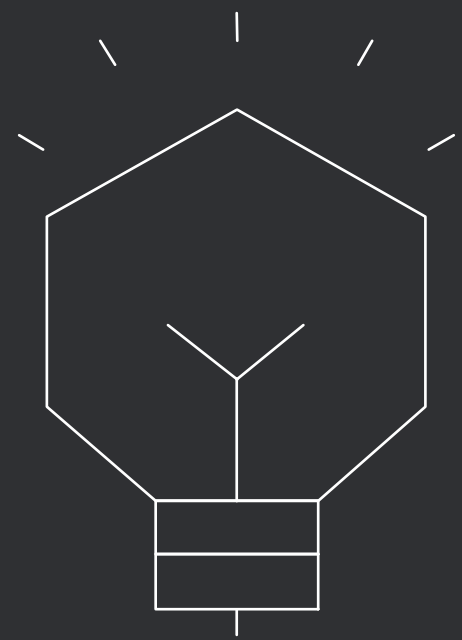
## Fluent assertions for java

```
assertThat(list).contains("abc");  
|   ->  
java.lang.AssertionError:  
Expecting:  
  <[]>  
to contain:  
  <["abc"]>  
but could not find:  
  <["abc"]>
```

# Unit Test Simplification







**Simplify** -> **Mock It!**

mockito



## Mockito: mocking method call

```
@Test
public void test_mocking_method_call()
{
    List mockedListLocal = mock(List.class);

    when(mockedListLocal.size()).thenReturn(10);
    when(mockedListLocal.get(anyInt())).thenReturn(100); // advanced feature with argumentMatcher

    assertEquals( expected: 10, mockedListLocal.size());
    assertEquals( expected: 100, mockedListLocal.get(123));

    // verify that method size() iz called
    verify(mockedListLocal).size();
    // all matchers like any() can be found in class ArgumentMatchers
    // all verification modes like times(0) can be found in class VerificationModeFactory
    verify(mockedListLocal, times( wantedNumberOfInvocations: 1)).size();
}
```

## Mockito: mocking method with different return values on consecutive calls

```
@Test
public void test_consecutive_method_calls_with_different_return_values()
{
    when(mockedList.get(0))
        .thenReturn("one", "two", "three");

    assertEquals(mockedList.get(0), "one");
    assertEquals(mockedList.get(0), "two");
    assertEquals(mockedList.get(0), "three");
}
```

## Mockito: using SPY

```
@Test
public void test_spying_on_real_objects()
{
    List list = new LinkedList();
    List spy = spy(list);

    //optionally, you can stub out some methods:
    // WARNING: always use syntax starting with doReturn while using spy to prevent call to real method, e.g. inside when(get(0));
    doReturn( toBeReturned: 100 ).when(spy).size();

    //using the spy calls *real* methods
    spy.add("one");
    spy.add("two");

    //prints "one" - the first element of a list
    System.out.println(spy.get(0));

    //size() method was stubbed - 100 is printed
    System.out.println(spy.size());
    assertEquals(spy.size(), actual: 100);
}
```

## Mockito: using SPY

```
@Test
public void test_spying_on_real_objects()
{
    List list = new LinkedList();
    List spy = spy(list);

    //optionally, you can stub out some methods:
    // WARNING: always use syntax starting with doReturn while using spy to prevent call to real method, e.g. inside when(get(0));
    doReturn(toBeReturned: 100).when(spy).size();

    //using the spy calls *real* methods
    spy.add("one");
    spy.add("two");

    //prints "one" - the first element of a list
    System.out.println(spy.get(0));

    //size() method was stubbed - 100 is printed
    System.out.println(spy.size());
    assertEquals(spy.size(), actual: 100);
}
```



Successfully mocked objects!

**How close to 100% code coverage?**



PowerMock



## Powermock: Mock System Classes

```
private void mockURLDecoder()
{
    PowerMockito.mockStatic(URLDecoder.class);
    try
    {
        PowerMockito.when(URLDecoder.decode(anyString(), eq(StandardCharsets.UTF_8.name()))).
            thenThrow(new UnsupportedOperationException("exception"));
    }
    catch (final UnsupportedOperationException e)
    {
        // log exception
    }
}
```

## Powermock: Mock Static Methods

```
@RunWith(PowerMockRunner.class)
@PrepareForTest(Static.class)
public class YourTestCase {
    @Test
    public void testMethodThatCallsStaticMethod() {
        //given
        // mock all the static methods in a class called "Static"
        PowerMockito.mockStatic(Static.class);
        // use Mockito to set up your expectation
        PowerMockito.when(Static.firstStaticMethod(param)).thenReturn(value);
        PowerMockito.when(Static.secondStaticMethod()).thenReturn(123);

        // when
        // execute your test
        classCallStaticMethodObj.execute();

        // then
        // Different from Mockito, always use PowerMockito.verifyStatic(Class) first
        // to start verifying behavior
        PowerMockito.verifyStatic(Static.class);
    }
}
```


## Powermock: Mock Constructor Call

```
public class ClassUnderTest {
    // ...

    private void method1() {
        //...
        Something smth = new Something("arg1"); // this line is being mocked
        //...
    }
}

@PrepareForTest(ClassUnderTest.class) // IMPORTANT: not the class that is being mocked but the one under test!
public class ClassTest {
    //... test methods

    public void customMethod() {
        //...
        PowerMockito.whenNew(Something.class).withArguments(argument).thenReturn(mockSomething);
        //...
    }
}
```



Motivation: Constructor/method loads dll file or accesses network which makes code "untestable".

Supressing:

- constructor of a superclass / own constructor and instantiating a class
- Removing static initializer for the class
- specific method / field initialization



## Powermock: Bypass Encapsulation

```
// getting value of private field
Whitebox.getInternalState(objectFromWhichFieldIsFetched, A.class);
// B.class defines field type, A.class defines in which class field is defined
Whitebox.getInternalState(objectFromWhichFieldIsFetched, fieldName: "fieldName", B.class, A.class);

// injecting value in private field
Whitebox.setInternalState(objectInWhichMockIsBeingInserted, "fieldName", fieldValueMock);
// finding correct field is based on fieldValueMock type
Whitebox.setInternalState(objectInWhichMockIsBeingInserted, fieldValueMock);

// invoking private method
Whitebox.invokeMethod(myInstance, param1, param2);
```

Motivation: automate common  
mocking tasks – **Mock Policy**

OOTB Mock Policy:

- `@MockPolicy(Slf4jMockPolicy.class)` – removes unnecessary logging, improves test/build performance and readability







# Documentation Generation

A black and white photograph of a young child, possibly a toddler, with a wide-eyed, open-mouthed expression of surprise or awe. The child is wearing a light-colored long-sleeved shirt under a dark, textured vest. They are holding an open book in front of them with both hands. The background is dark and out of focus.

**Documentation created from JUnit?**



**JGiven<sup>o</sup>**



## Why JGiven?

- **Java** code using a fluent API
- **JUnit** or TestNG support
- **Modular** way of writing Scenarios
- **Reports** are generated for domain experts

## JGiven

```
@Test
public void a_pancake_can_be_fried_out_of_an_egg_milk_and_flour() {
    given().an_egg().
        and().some_milk().
        and().the_ingredient( "flour" );

    when().the_cook_mangles_everything_to_a_dough().
        and().the_cook_fries_the_dough_in_a_pan();

    then().the_resulting_meal_is_a_pancake();
}
```

Scenario: a pancake can be fried out of an egg milk and flour

Given an egg  
And some milk  
And the ingredient flour  
When the cook mangles everything to a dough  
And the cook fries the dough in a pan  
Then the resulting meal is a pan cake

## Stages Example

```
import com.tngtech.jgiven.Stage;

public class GivenSomeState extends Stage<GivenSomeState> {
    public GivenSomeState some_state() {
        return self();
    }
}

import com.tngtech.jgiven.Stage;

public class WhenSomeAction extends Stage<WhenSomeAction> {
    public WhenSomeAction some_action() {
        return self();
    }
}

import com.tngtech.jgiven.Stage;

public class ThenSomeOutcome extends Stage<ThenSomeOutcome> {
    public ThenSomeOutcome some_outcome() {
        return self();
    }
}
```

## Scenario Example

```
public class UsingRulesTest {

    @ClassRule
    public static final JGivenClassRule writerRule = new JGivenClassRule();

    @Rule
    public final JGivenMethodRule scenarioRule = new JGivenMethodRule();

    @ScenarioStage
    GivenSomeState someState;

    @ScenarioStage
    WhenSomeAction someAction;

    @ScenarioStage
    ThenSomeOutcome someOutcome;

    @Test
    public void something_should_happen() {
        someState.given().some_state();
        someAction.when().some_action();
        someOutcome.then().some_outcome();
    }
}
```

A grayscale photograph of a hand raised in a classroom. The hand is positioned on the right side of the frame, with fingers spread. The background is blurred, showing other people and papers on a wall. The lighting is soft, creating a bokeh effect in the upper left corner.

That's it.

**Questions?**