By the power of mockito…

Test it, but do it the smart way…

CSS Versicherung

Timetable

1. Why drink it?
2. How do I drink it?
3. How do I «mockito»?
4. Conclusion

“We decided during the main conference that we should use JUnit 4 and Mockito because we think they are the future of TDD and mocking in Java”

Daniel Terhorst-North, originator of Behavior-Driven Development

Why drink it?

- Tastes really good
- Beautiful tests due to clean and simple API
- No hangover
- Best mocking framework for Java
- #4 Java library

How do I drink it?

- Do not mock types you don’t own
- Don’t mock value objects
- Don’t mock everything
- Show love with your tests!

How do I «mockito»?

- Creation
- Behavior
- Specification
- Spy
- Injection
- Verification

Create

- With mock()

```
TimeService timeService = mock(TimeService.class);
```

- With @Mock

```
@Mock
private TimeService timeService;
```

Behave

- With answer

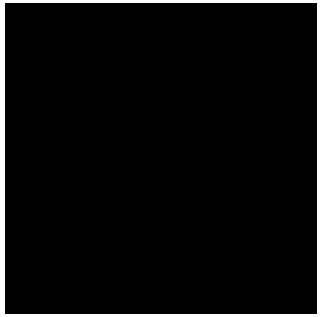
```
final ArgumentCaptor<IInstrumentEventListener> listenerCaptor = ArgumentCaptor.forClass(IInstrumentEventListener.class);
verify(instrumentMock).setEventListener(listenerCaptor.capture());
doAnswer(invocation -> {
    final String task = invocation.getArgumentAt( index: 0, String.class);
    listenerCaptor.getValue().finished(task);
    return null;
}).when(instrumentMock).execute(TASK);
```

- With settings

```
@Mock(answer = Answers.RETURNS_SMART_NULLS)
private ShareService shareService;

mock(ShareService.class, withSettings().defaultAnswer(RETURNS_SMART_NULLS));
```

Specify



- With when/then

```
when(transactionService.getTransactions()).thenReturn(List.of(transaction));
when(priceService.getCurrentPrice(OLD_SCHOOL_WATERFALL_SOFTWARE_LTD)).thenReturn(5.75);
```

- With given/will (BDD aproach)

```
given(transactionService.getTransactions()).willReturn(List.of(transaction));
given(priceService.getCurrentPrice(OLD_SCHOOL_WATERFALL_SOFTWARE_LTD)).willReturn(5.75);
```

Spy

- With spy()

```
PortfolioService portfolioServiceSpy =
        spy(new PortfolioService(timeService, priceService, printerService, transactionService, shareService));
```

- With @Spy

```
@Spy
private PortfolioService portfolioService
        = new PortfolioService(timeService, priceService, printerService, transactionService, shareService);
```

- Why spy?
  - Call real methods
  - Stub method calls
  - Verify calls

Inject

- Automatically inject mock/spy fields annotated with @Spy or @Mock

```
private PortfolioService portfolioService
        = new PortfolioService(timeService, priceService, printerService, transactionService, shareService);
```

```
@InjectMocks
private PortfolioService portfolioService;
```

Verify

- With arguments

```
verify(printerService).print("company | shares | current price | current value | last operation\n");
```

- With matchers

```
verify(printerService).print(any());
```

- With captors

```
@Captor
private ArgumentCaptor<Transaction> argumentCaptor;

final ArgumentCaptor<Transaction> argumentCaptor = ArgumentCaptor.forClass(Transaction.class);
verify(transactionService).store(argumentCaptor.capture());
assertEquals(transaction, argumentCaptor.getValue());
```

Important

- Annotations only work if initMocks() is run:

```
@BeforeEach
void setUp() {
    MockitoAnnotations.initMocks( testClass: this);
}
```

- Or the test class is extended with MockitoExtension:

```
@ExtendWith(MockitoExtension.class)
class PortfolioServiceShould {
```

Conclusion

- Great framework
- Easy to learn – hard to master
- Use BDDMockito for Behavior-Driven Development

Thank you!

- References:
  - «Mockito framework site» (see https://site.mockito.org/)
  - «Agile Technical Practices Distilled» by Pedro Moreira Santos, Marco Consolaro & Alessandro Di Gioia
- Kontakt:
  - luca.zangger@css.ch