



---

# Advanced Mocking mit PowerMock

**Michael Inden**

---



- **Motivation**
  - **PowerMock im Überblick**
    - Statische Methoden mocken
    - Finale Methoden mocken
    - Private Methoden mocken
  - **Weitere Möglichkeiten**
    - Ungewünschte Aufrufe verhindern
-



---

# Motivation

---



```
public class FileUtils
{
    public boolean checkExistance(File file)
    {
        return file.exists();
    }
}
```

## Beispiel rein mit Mockito

---



```
public class FileUtilsTest
{
    @Test
    public void checkExistence()
    {
        // ARRANGE
        File file = Mockito.mock(File.class);
        Mockito.when(file.exists()).thenReturn(true);

        // ACT
        FileUtils demo = new FileUtils();
        boolean exists = demo.checkExistence(file);

        // ASSERT
        assertTrue(exists);
    }
}
```

## Beispiel: Kleine Änderung, große Auswirkung für Testbarkeit

---



```
public class FileUtilsV2
{
    public boolean checkExistance(String path)
    {
        final File file = new File(path);
        return file.exists();
    }
}
```



# Wie schreibe ich den Test denn nun? Wie komme ich an das neu erzeugte File-Objekt???

```
public boolean checkExistance(String path)
{
    final File file = new File(path);
    return file.exists();
}
```



---

# PowerMock im Überblick

---





- PowerMock ist ein Framework zur Ergänzung verschiedener Mocking-Tools
  - PowerMock gibt es als Erweiterung zu Mockito und auch EasyMock
  - PowerMock bietet noch mehr Möglichkeiten als die Standard-Tools und erlaubt das Mocking von
    - Statischen Methoden
    - Finalen Methoden und Klassen
    - Privaten Methoden
    - Konstruktoren
    - Entfernen des Aufrufs von statischen Initializer
    - Entfernen des Aufrufs von Konstruktoren, Methoden usw.
  - **PowerMock greift stark in existierenden Code ein**
  - PowerMock nutzt einen eigenen ClassLoader und ByteCode-Manipulation
-

# PowerMock Maven Dependencies

---



```
<!-- https://mvnrepository.com/artifact/org.powermock/powermock-core -->
```

```
<dependency>  
  <groupId>org.powermock</groupId>  
  <artifactId>powermock-core</artifactId>  
  <version>2.0.9</version>  
  <scope>test</scope>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.powermock/powermock-api-mockito2 -->
```

```
<dependency>  
  <groupId>org.powermock</groupId>  
  <artifactId>powermock-api-mockito2</artifactId>  
  <version>2.0.9</version>  
  <scope>test</scope>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.powermock/powermock-module-junit4 -->
```

```
<dependency>  
  <groupId>org.powermock</groupId>  
  <artifactId>powermock-module-junit4</artifactId>  
  <version>2.0.9</version>  
  <scope>test</scope>  
</dependency>
```

---

# Initiales Beispiel

---



```
public class Calculator
{
    public int add(int a, int b)
    {
        return CalculatorService.add(a, b);
    }
}

public final class CalculatorService
{
    public static final int add(int a, int b)
    {
        return a + b;
    }

    private CalculatorService()
    {
    }
}
}
```

---

# Wichtige Schritte



- 1) Speziellen Testrunner und zu mockende Klassen angeben:

Wird speziell geladen

```
@RunWith(PowerMockRunner.class)  
@PrepareForTest(CalculatorService.class)
```

- 2) Zu mockende Klassen spezifizieren

```
PowerMockito.mockStatic(CalculatorService.class);  
PowerMockito.mock(CalculatorService.class);
```

=> finale Methoden

- 3) Erwartetes Verhalten vorgeben

```
PowerMockito.when(CalculatorService.add(1, 1)).thenReturn(2);
```

# Initiales Beispiel




```
@RunWith(PowerMockRunner.class)
@PrepareForTest(CalculatorService.class)
public class CalculatorTest
{
    @Test
    public void multiply_instead_of_add()
    {
        // ARRANGE
        final Calculator calc = new Calculator();
        PowerMockito.mockStatic(CalculatorService.class);
        PowerMockito.when(CalculatorService.add(1, 1)).thenReturn(1);
        PowerMockito.when(CalculatorService.add(5, 2)).thenReturn(10);

        // ACT
        int result1 = calc.add(1, 1);
        int result2 = calc.add(5, 2);

        // ASSERT
        assertEquals(1, result1);
        assertEquals(10, result2);
    }
}
```

Runs: 1/1    ✗ Errors: 0    ✕ Failures: 0

▼  CalculatorTest [Runner: JUnit 5] (0.781 s)

 multiply\_instead\_of\_add (0.781 s)

## Initiales Beispiel – Besonderheit bei doThrow()



```
@RunWith(PowerMockRunner.class)
@PrepareForTest(CalculatorService.class)
public class CalculatorTest
{
    ...

    @Test
    public void special_handling_of_invalid_input()
    {
        // ARRANGE
        final Calculator calc = new Calculator();
        PowerMockito.mockStatic(CalculatorService.class);
        // Unfinished stubbing
        // PowerMockito.doThrow(new ArithmeticException()).
        //                      when(CollaboratorWithStaticMethods.add(-1, -1));

        PowerMockito.when(CalculatorService.add(-1, -1)).thenThrow(new ArithmeticException());

        // ACT & ASSERT
        assertThrows(ArithmeticException.class, () -> calc.add(-1, -1));
    }
}
```

# Initiales Beispiel – Besonderheit bei doThrow()



```
@RunWith(PowerMockRunner.class)
@PrepareForTest(CalculatorService.class)
public class CalculatorTest
{
    ...

    @Test
    public void special_handling_of_invalid_input()
    {
        // ARRANGE
        final Calculator calc = new Calculator();
        PowerMockito.mockStatic(CalculatorService.class);
        PowerMockito.doThrow(new ArithmeticException()).when(CalculatorService.class);
        CalculatorService.add(-1, -1);

        // ACT & ASSERT
        assertThrows(ArithmeticException.class, () -> calc.add(-1, -1));
    }
}
```



**Wie prüfe ich, ob die  
statische Methode  
aufgerufen wurde?**



# Initiales Beispiel – Besonderheit bei doThrow()



```
@RunWith(PowerMockRunner.class)
@PrepareForTest(CalculatorService.class)
public class CalculatorTest
{
    ...

    @Test
    public void verify_static_method_is_called()
    {
        // ARRANGE
        final Calculator calc = new Calculator();
        PowerMockito.mockStatic(CalculatorService.class);
        PowerMockito.when(CalculatorService.add(1, 1)).thenReturn(1);
        PowerMockito.when(CalculatorService.add(5, 2)).thenReturn(10);

        // ACT
        calc.add(1, 1);

        // ASSERT / VERIFY
        PowerMockito.verifyStatic(CalculatorService.class);
        CalculatorService.add(1, 1);
    }
}
```



**Jetzt wissen wir einiges,  
aber wie können wir das  
für die FileUtilsV2 und  
den Konstruktor nutzen?**

## RECAP: Beispiel rein mit Mockito (Ausgangslage)



```
public class FileUtils
{
    public boolean checkExistance(File file)
    {
        return file.exists();
    }
}

public class FileUtilsTest
{
    @Test
    public void checkExistance()
    {
        File file = Mockito.mock(File.class);
        Mockito.when(file.exists()).thenReturn(true);

        FileUtils demo = new FileUtils();
        boolean exists = demo.checkExistance(file);

        assertTrue(exists);
    }
}
```

## Beispiel: Kleine Änderung, große Auswirkung für Testbarkeit

---



```
public class FileUtilsV2
{
    public boolean checkExistence(String path)
    {
        final File file = new File(path);
        return file.exists();
    }
}
```

## Beispiel – Einklinken in den Konstruktor



```
@RunWith(PowerMockRunner.class)
public class FileUtilsV2Test
{
    @Test
    @PrepareForTest(FileUtilsV2.class)
    public void testCallArgumentInstance() throws Exception
    {
        File file = PowerMockito.mock(File.class);
        PowerMockito.whenNew(File.class).withArguments("TESTFILE").thenReturn(file);
        PowerMockito.when(file.exists()).thenReturn(true);

        // ACT
        FileUtilsV2 demo = new FileUtilsV2();
        boolean exists = demo.checkExistence("TESTFILE");

        // ASSERT
        assertTrue(exists);
    }
}
```



---

# Weitere Möglichkeiten

# Ungewünschte Aufrufe verhindern

---



- Statische Initializer verhindern

```
@SuppressWarnings("»my.package.ClassWithEvilStaticInitializer")
```

- Konstruktoraufruf verhindern

```
suppress(constructor(EvilParent.class))
```

```
Whitebox.newInstance(ClassWithEvilConstructor.class)
```

- Methodenaufrufe verhindern

```
suppress(method(ClassWithEvilMethod.class, "methodName"))
```

---

# Ungewünschte Aufrufe verhindern



```
public class StaticInitializerExample
{
    static
    {
        DBUtils.initialize();
        // weitere Calls ...
    }

    public StaticInitializerExample()
    {
    }
}

public class DBUtils
{
    public static void initialize()
    {
        // for demo purposes
        throw new IllegalStateException();
    }
}
```



# Ungewünschte Aufrufe verhindern



```
@RunWith(PowerMockRunner.class)
public class StaticInitializerExampleTest
{
    @PrepareForTest(StaticInitializerExample.class)
    @SuppressStaticInitializationFor("a_powermockito_intro_examples.StaticInitializerExample")
    @Test
    public void testWithoutStaticInitializerd()
    {
        StaticInitializerExample instance = new StaticInitializerExample();
    }
}
```

Nicht nötig

# Ungewünschte Methodenaufrufe verhindern

---



```
import static org.powermock.api.support.membermodification.MemberMatcher.method;  
import static org.powermock.api.support.membermodification.MemberModifier.suppress;
```

```
@PrepareForTest(StaticInitializerExample.class)  
@Test  
public void testWithoutFaultyMethodCall()  
{  
    suppress(method(DBUtils.class, "initialize"));  
  
    StaticInitializerExample instance = new StaticInitializerExample();  
}
```

# Ungewünschte Konstruktoraufrufe verhindern

---



```
public class SuperClass
{
    public SuperClass()
    {
        throw new IllegalArgumentException();
    }
}
```

```
public class SubClass extends SuperClass
{
    public String someMethod()
    {
        return "RESULT FROM CHILD";
    }
}
```


---

# Ungewünschte Konstruktoraufrufe verhindern



```
import static org.powermock.api.support.membermodification.MemberMatcher.constructor;  
import static org.powermock.api.support.membermodification.MemberModifier.suppress;
```

```
@RunWith(PowerMockRunner.class)  
@PrepareForTest({ SuperClass.class, SubClass.class })  
public class SuppressSuperConstructorCallTest  
{  
    @Test  
    public void testSuppressSuperConstructorCall()  
    {  
        suppress(constructor(SuperClass.class));  
  
        String result = new SubClass().someMethod();  
  
        assertEquals("RESULT FROM CHILD", result);  
    }  
}
```



In aktuellen  
Versionen  
nicht mehr  
möglich!



---

# DEMO

---



- PowerMockito ist ein wirklich mächtiges Tool
- Gerade im Kontext von Legacy Code manchmal die letzte Möglichkeit, Testbarkeit herzustellen
- Hilfreich, um statische oder finale Methoden testbar zu machen
- PowerMockito noch nicht 100% JUnit 5 kompatibel
- Merkwürdige Probleme in Kombination Mockito / PowerMockito und Java  $\geq 9$

## Bedenkenswertes

- Sollte man private Methoden wirklich redefinieren?
  - Sollte man wirklich statische Initializer-Blöcke / Konstruktoren überspringen?
    - Bei Legacy Code manchmal ja, bei gutem Design eher nein
  - **Oftmals deutet der Einsatz von PowerMockito darauf hin, dass das Design verbessert werden sollte**
-





- <https://github.com/powermock/powermock>
  - <https://www.baeldung.com/intro-to-powermock>
  - <https://www.javaindeed.com/3-best-practices-to-test-a-code-that-calls-static-methods/>
  - <https://programmer.help/blogs/powermockito-use-details.html>
  - <https://dev.to/aldok/how-to-use-powermockito-whennew-16m2>
  - <https://www.javatpoint.com/mockito-powermock>
  - <https://reliablesoftwareblog.wordpress.com/2016/03/28/mocking-a-constructor-a-unit-test-for-a-factory-method/>
-





---

# Thank You

---