
Write Unit Tests with and Hamcrest

# Content

# 01

## Intro

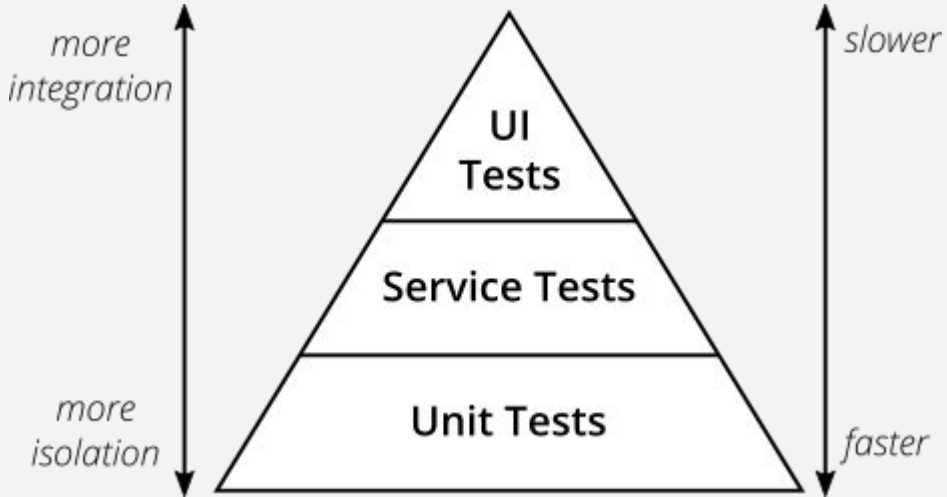
# Intro

**JUnit** - testing framework that helps with writing and running your unit tests. Supported by all IDE, build tools (ant, maven, gradle) and popular frameworks (spring)
**Junit 5** - (sept 2017, java 8+) is not one jar anymore as it was Junit 4 (**JUnit 5 =** ***JUnit Platform*** **+** ***JUnit Jupiter*** **+** ***JUnit Vintage*** )

**Mockito** - (or any other mocking tool) is a mock library that you specifically use to efficiently write certain kind of tests.

**PowerMock** - mock library for some powerfull features (mocking of static methods, final classes, constructors, private methods, removal of static initializers and more)

**Hamcrest** - help library that assists writing software tests in the Java programming language.




# 02

## Mocking with Mockito

# Mocking ...

**Mocking is a testing technique where real components are replaced with objects that have a predefined behavior (mock objects) only for the test/tests that have been created for.**
**There are several scenarios where we should use mocks:**

- **When we want to test a component that depends on other component, but which is not yet developed**
- **When the real component performs slow operations,**
- **When there are infrastructure concerns that would make impossible the testing**

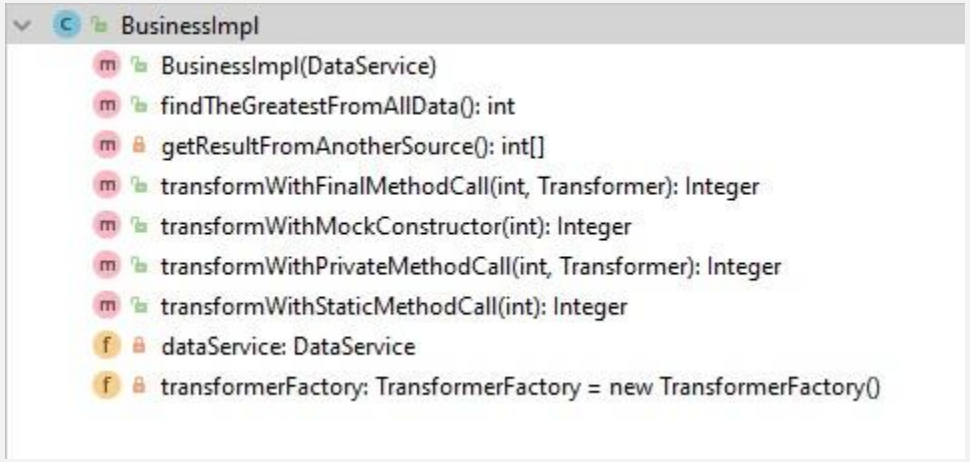
seavus

# Importing

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>${mockito-core.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.powermock</groupId>
    <artifactId>powermock-module-junit4</artifactId>
    <version>${powermock-module-junit4.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.powermock</groupId>
    <artifactId>powermock-api-mockito2</artifactId>
    <version>${powermock-api-mockito.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.powermock</groupId>
    <artifactId>powermock-api-mockito-common</artifactId>
    <version>${powermock-api-mockito-common.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-all</artifactId>
    <version>${hamcrest-all.version}</version>
    <scope>test</scope>
</dependency>
```

# Implementing

We have a class com.seavus.mockito.presentation.BusinessImpl




And we need to write unit tests….

# Without mocking

In test package we need to create “stubs” which will act as replacement for real objects.

```
public class DataServiceStub implements DataService {


    @Override
    public int[] retrieveDataFromSomewhere() {
        return new int[]{100, 500, 200};
    }


    @Override
    public void doSomethingElse(){
```

And use them in test as:

```
package com.seavus.mockito.presentation;


import static org.junit.Assert.assertEquals;
import org.junit.Test;


public class BusinessImplStubTest {
    @Test
    public void testFindTheGreatestFromAllData() {
        BusinessImpl businessImpl = new BusinessImpl(new DataServiceStub());
        int result = businessImpl.findTheGreatestFromAllData();
        assertEquals(500, result);
    }
}
```

Problems with this stub:

- How do I get DataServiceStub to return different data for different scenarios?
- Everytime DataService interface is updated with new methods, the DataServiceStub implementations should be updated.

# With mocking

In test package we DON”T need to create “stubs” which will act as replacement for real objects. Mocking library does it.

```
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;


public class BusinessMockTest {


    @Test
    public void testFindTheGreatestFromAllData() {
        DataService dataServiceMock = mock(DataService.class); //creation of mock object
        when(dataServiceMock.retrieveDataFromSomewhere()).thenReturn(new int[] { 100, 200, 500 });//stub
        BusinessImpl businessImpl = new BusinessImpl(dataServiceMock);
        int result = businessImpl.findTheGreatestFromAllData();
        assertEquals(500, result);
        verify(dataServiceMock, times(1)).retrieveDataFromSomewhere();
    }


    @Test
    public void testFindTheGreatestFromAllData_ForOneValue() {
        DataService dataServiceMock = mock(DataService.class); //creation of mock object
        when(dataServiceMock.retrieveDataFromSomewhere()).thenReturn(new int[] { 15 });
        BusinessImpl businessImpl = new BusinessImpl(dataServiceMock);
        int result = businessImpl.findTheGreatestFromAllData();
        assertEquals(15, result);
        verify(dataServiceMock, times(1)).retrieveDataFromSomewhere();
    }
```

Problem:

- Creation of mock object in each test.

seavus

# With mocking and annotations

Test code is more clear by using annotations.

```java
@RunWith(MockitoJUnitRunner.class)
public class BusinessMockAnnotationsTest {

    @Mock
    DataService dataServiceMock;

    @InjectMocks
    BusinessImpl businessImpl;

    @Test
    public void testFindTheGreatestFromAllData() {
        //given
        int[] array = new int[] { 100, 500, 300 };
        when(dataServiceMock.retrieveDataFromSomewhere()).thenReturn(array);
        //when
        int result = businessImpl.findTheGreatestFromAllData();
        //then
        assertEquals(500, result);
        verify(dataServiceMock).retrieveDataFromSomewhere(); //or
        verify(dataServiceMock, times(1)).retrieveDataFromSomewhere();
        verify(dataServiceMock, atLeastOnce()).doSomethingElse(500);//or
        verify(dataServiceMock, atLeastOnce()).doSomethingElse(anyInt());//or to use captor
    }
```

```java
public class DataServiceImpl implements DataService {

@Override
public int[] retrieveDataFromSomewhere() {
    System.out.println("Calling external source to obtain data");
    …….
}
```

```java
public class BusinessImpl {

    private DataService dataService;

    public int findTheGreatestFromAllData() {
        int[] data = dataService.retrieveDataFromSomewhere();
        int min = Integer.MIN_VALUE;

        for (int value : data) {
            if (value > min) {
                min = value;
            }
        }
        if(min > 400) {
            dataService.doSomethingElse(min);
        }
        //do something else (static method call)
        return min;
    }
```

seavus

# Argument matchers

- **ArgumentMatchers allows us flexible verification or stubbing.**

```
when(mockFoo.someMethod("someString", 5, somelist).thenReturn(true);
when(mockFoo.someMethod(anyString(), anyInt(), anyList())).thenReturn(true);
```

- **In case of a method has more than one argument, it isn't possible to use ArgumentMatchers for only some of the arguments. Mockito requires you to provide all arguments either by matchers or by exact values.**
- **We can't use them as a return value, an exact value is required when stubbing calls.**
- **We can't use argument matchers outside of verification or stubbing.**

```
verify(mockFoo, atLeast(0)).someMethod(anyString(), anyInt(), any(List.class));
```

- **Custom Argument Matcher vs. ArgumentCaptor**
- **Both techniques custom argument matchers and ArgumentCaptor can be used for making sure certain arguments were passed to mocks.**
- **ArgumentCaptor may be a better fit if we need it to assert on argument values to complete verification or our custom argument matcher is not likely to be reused.**
- **Custom argument matchers via ArgumentMatcher are usually better for stubbing.**


# 03

## Powermock library

seavus®

# What is Powermock?

- **PowerMock is a library that extends other mock libraries such as Mockito and EasyMock with more powerful capabilities.**
- **PowerMock uses a custom classloader and bytecode manipulation to enable mocking of: static methods, constructors, final classes and methods, private methods, removal of static initializers and more.**

- ☑ **Mocking static methods**
- ☑ **Mocking final methods**
- ☑ **Mocking private methods**
- ☑ **Mocking constructors**

seavus

# Mocking static method with Powermock

- Use the @RunWith(PowerMockRunner.class) annotation at the class-level of the test case.
- Use the @PrepareForTest(ClassThatContainsStaticMethod.class) annotation at the class-level of the test case.
- Use PowerMockito.mockStatic(ClassThatContainsStaticMethod.class) to mock all methods of this class.
- Use PowerMockito.verify(ClassThatContainsStaticMethod.class) to change the class to verify mode.

seavus

# Mocking final method with Powermock

- Use the @RunWith(PowerMockRunner.class) annotation at the class-level of the test case.
- Use the @PrepareForTest(ClassWithFinal.class) annotation at the class-level of the test case.
- Use PowerMockito.mock(ClassWithFinal.class) to create a mock object for all methods of this class (let's call it mockObject).
- Use PowerMockito.*verify*(mockObject) to change the mock object to verify mode.

seavus

# Mocking private method with Powermock

- Use the @RunWith(PowerMockRunner.class) annotation at the class-level of the test case.

- Use the @PrepareForTest(ClassWithPrivateMethod.class) annotation at the class-level of the test case.
- Use the PowerMockito.*doReturn*(returnValue).when(spyObject, "nameOfTheMethodToMock", args) on method level.

- Use PowerMockito.*verifyPrivate*(spyObject).invoke("nameOfTheMethodToMock", args) to change the mock object to verify mode.

seavus

# Mocking constructor with Powermock

- Use the @RunWith(PowerMockRunner.class) annotation at the class-level of the test case.
- Use the @PrepareForTest(FactoryClass.class) annotation at the class- level of the test case.
- Use Mockito.mock(ClassThatNeedsToBeInstantiated.class,) to create a mock object  (let's call it mockObject).

- Use PowerMockito.*whenNew*(ClassThatNeedsToBeInstantiated.**class**).withNoArguments().thenReturn(mockObject) to return mock object instead of  new class instance.
- Use PowerMockito.*whenNew*(ClassThatNeedsToBeInstantiated.**class**).withArguments(arg1, arg2).thenReturn(mockObject) to return mock object instead of  new class instance.

- Use PowerMockito.*verifyNew*(Transformer.**class**).withNoArguments() to verify that called new instance creation.
- Use PowerMockito.*verifyNew*(Transformer.**class**).withArguments(arg1, arg 2) to verify that called new instance creation.

# 04

## Hamcrest

seavus

# Why Hamcrest?

- What's wrong with assertEquals?
  assertEquals(someString, someOtherString)
  assertEquals( cat.getName(), otherCat.getName())
- What about collections?
  - assertEquals(someKitten, cat.getKittens().iterator().next())
    This works if our kitten is the first element in the collection, but what about asserting that the kitten exists anywhere in the collection?
    Well, we can do this:

```
boolean found = false;
for (Kitten kitten : cat.getKittens()) {
    if (kitten.equals(someKitten)) {
        found = true;
    }
}
assertTrue(found);
```

- Or, by using Hamcrest:
  assertThat(cat.getKittens(), hasItem(someKitten))
  We have created Matcher on Iterable<Kitten> that accepts a Matcher<Kitten>

# Kind of Hamcrest matchers

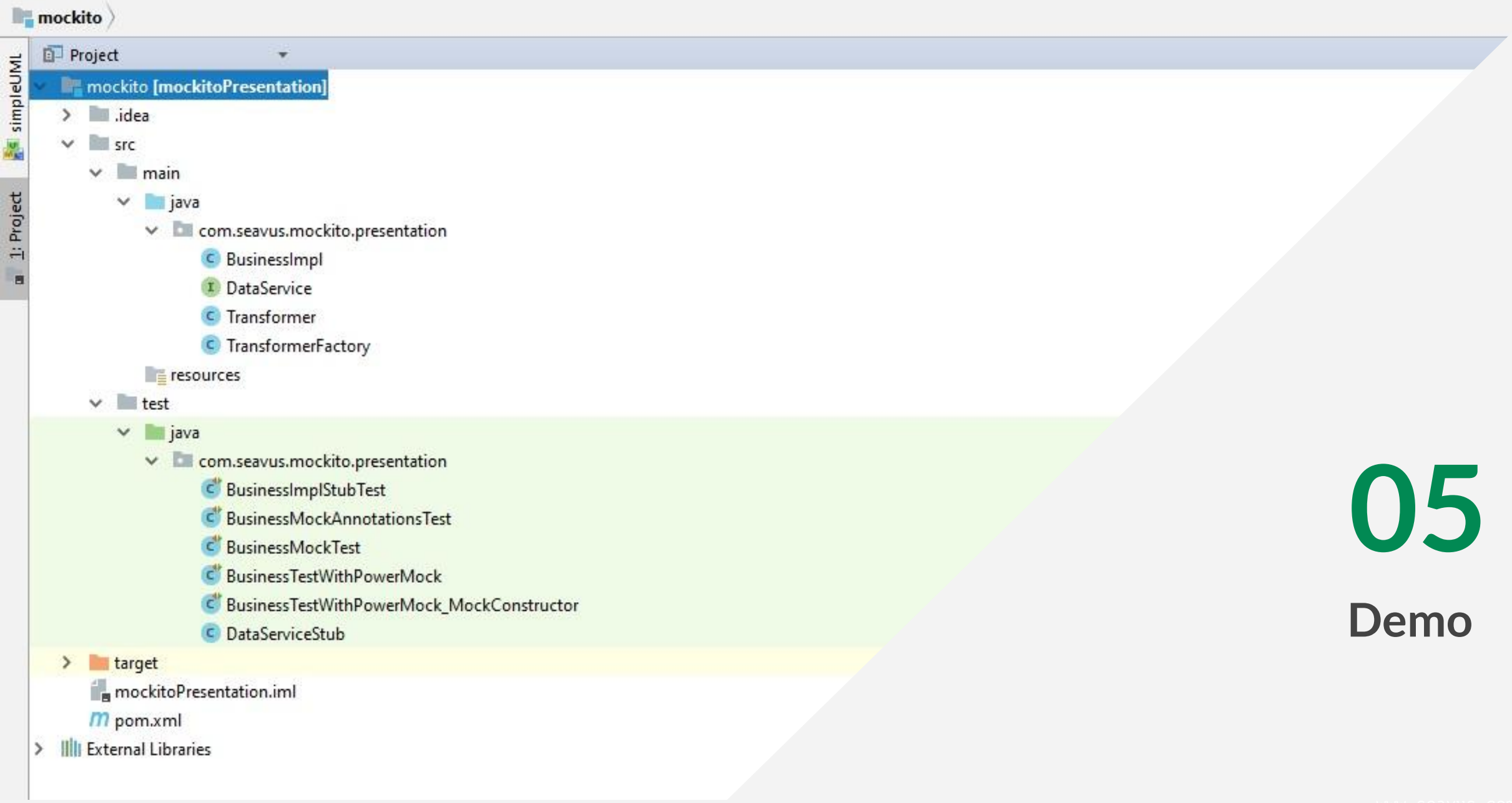
- Core matchers (startsWith, endsWith, is, isNull, everyItem)
- Logical matchers (allOf, anyOf, anything etc..)
- Object matchers (instanceOf, typeCompatibleWith, nullValue)
- Beans matchers (hasProperty, samePropertyValuesAs, )
- Collection matchers (hasSize, hasItem, hasItems, contains, containsInAnyOrder etc...)
- Number matchers (lessThan, greaterThan, greaterThanOrEqualTo, lessThanOrEqualTo)
- Text matchers (isEmptyOrNullString, containsString, equalToIgnoringWhiteSpace etc.. )
- XML matcher (hasXPath)

Wide offer of custom argument matchers (on mvn repo, github…)

JSON path matcher (hasJsonPath,):

```
<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path-assert</artifactId>
    <version>2.2.0</version>
</dependency>

// Verify evaluation of JSON path
assertThat(json, hasJsonPath("$.message", equalTo("Hi there")));
```


mockito
Project
mockito [mockitoPresentation]
.idea
src
main
java
com.seavus.mockito.presentation
BusinessImpl
DataService
Transformer
TransformerFactory
resources
test
java
com.seavus.mockito.presentation
BusinessImplStubTest
BusinessMockAnnotationsTest
BusinessMockTest
BusinessTestWithPowerMock
BusinessTestWithPowerMock_MockConstructor
DataServiceStub
target
mockitoPresentation.iml
pom.xml
External Libraries
simpleUML
1: Project

# 05

## Demo

# Links

Presentation on google slide:
https://docs.google.com/presentation/d/1o8AmqIwSSNwa1v9KPM2xgTFOahi1pf82Q7gAQ7vJp8k/edit#slide=id.p1

Demo on github:
https://github.com/igiton/presentations.git

Use your smartphone to scan with QR code scanner app