Intensivkurs JUnit

Agenda

- **Vorstellung**
- Grundlagen
- JUnit Framework
- Mock-Objekte mit Mockito
- Fortgeschrittene Möglichkeiten
- Ausblick

Vorstellung

- Trainer: Michael Zöller
- Aus Hamburg
- Team Lead, Software Architekt, Entwickler
- Twitter: @zettsystems
- Xing: https://www.xing.com/profile/Michael_Zoeller3
- LinkedIn: https://www.linkedin.com/in/michael-z%C3%B6ller-579041256/
- Mail: michael2.zoeller@gmail.com

Agenda

- Vorstellung
- **Grundlagen**
- JUnit Framework
- Mock-Objekte mit Mockito
- Fortgeschrittene Möglichkeiten
- Ausblick

Grundlagen Testen

- Warum ist Testen notwendig?
- Wie kann Software getestet werden?
- Wie viel Testaufwand ist erforderlich?
- Unterschiedliche Teststufen
- Test-Driven-Development (TDD)

Berüchtigte Bugs

- Der erste „bug“ der Geschichte war tatsächlich eine Motte im Harvard Mark II PC am 9. September. Grace Murray Hopper hat ihn dokumentiert.
- Der Jahr-2000 Bug (Y2K): Um Speicher zu sparen wurde Jahreszahlen nur 2-stellig gespeichert…plötzlich brauchte man 4 Stellen.
- Die „Dhahran Rakete“ im 1. Golf-Krieg wurde nicht abgefangen, weil das Patriot System einen Computer-Fehler hatte…28 Soldaten starben.
- Der Mars Climate Orbiter flog Monate durch das All und wurde dann zerstört da das Navigationssystem mit dem metrischen System arbeitete und das Kontrollteam die Daten im imperialen System verwendete…
- Am 4. Juni 1996 explodierte die Ariane 5 Rakete kurz nach dem Start. Die Software der Rakete kam von Ariane 4 und konnte die 64-bit Daten nicht richtig verarbeiten…
- Amazon S3 Ausfall in den USA 2017 durch eine Kombination aus ungültigen Parametern für ein internes Tool, Admin-Fehler, nicht getesteten Wiederherstellroutinen, nicht aktualisierten Playbooks sowie außergewöhnlich hoher Last.

Warum ist Testen notwendig?

- Testen spart Geld, weil Fehler früh gefunden werden und nicht in Produktion gelangen
- Testen macht Produkte sicherer
- Testen erhöht die Produktqualität
- Testen macht die Wartbarkeit einfacher
- Testen gibt ein gutes Gefühl, dass die Software das Richtige macht
- Testen macht die Nutzer zufrieden
- Testen verbessert die Codequalität
- ...

Softwarequalität

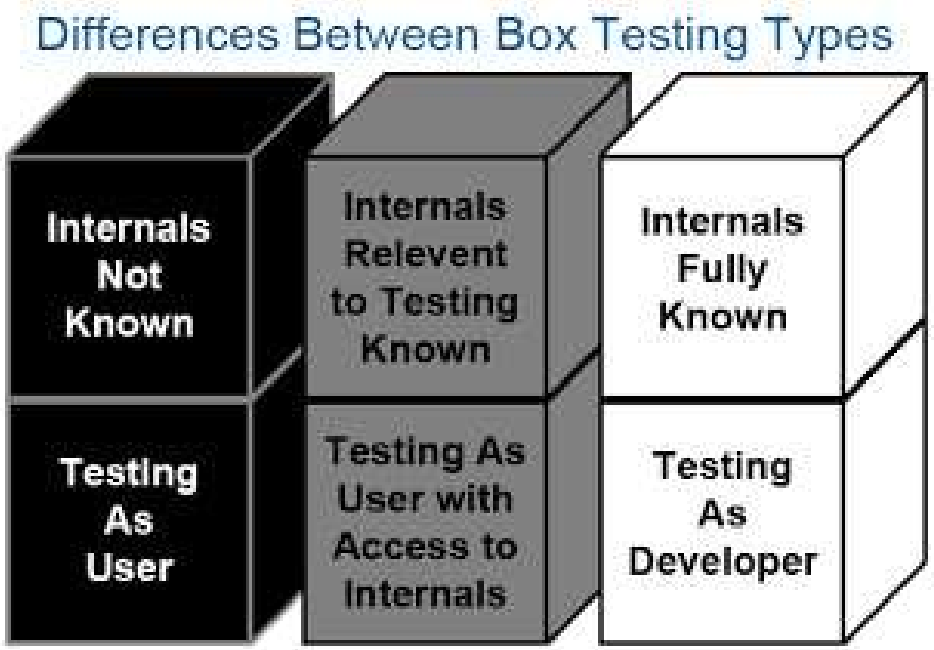
ISO/IEC 25010, https://www.inztitut.de/blog/glossar/iso-25010/

Wie kann Software getestet werden?

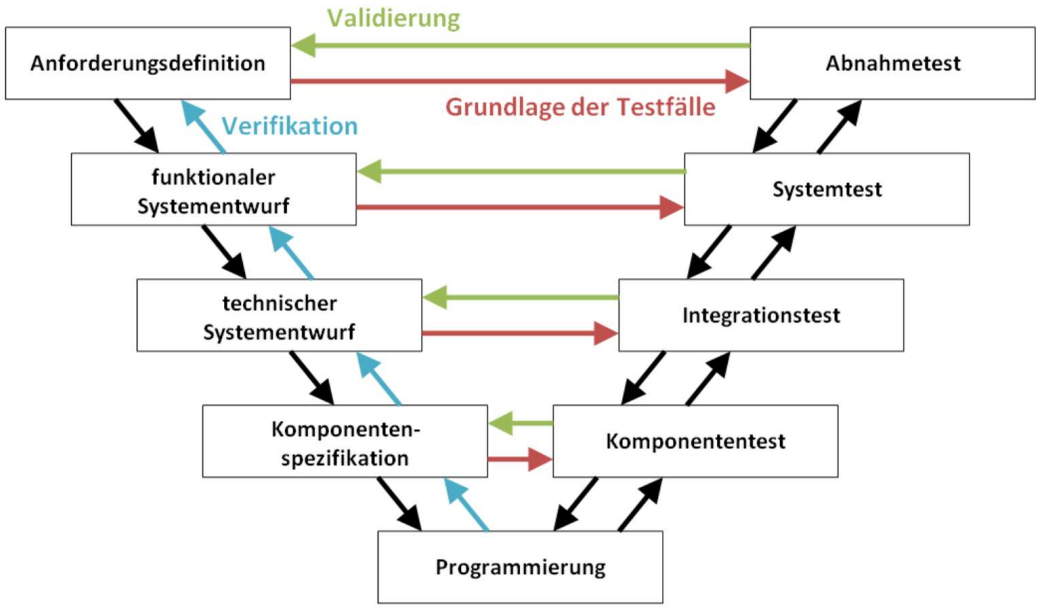
- Manuell: aufwändig, fehleranfällig, langweilig
- Automatisiert: deutlich besser!

- --> die meisten Tests sollten automatisiert erfolgen, nur explorative Tests oder wenn notwendig Abnahmetests erfolgen manuell

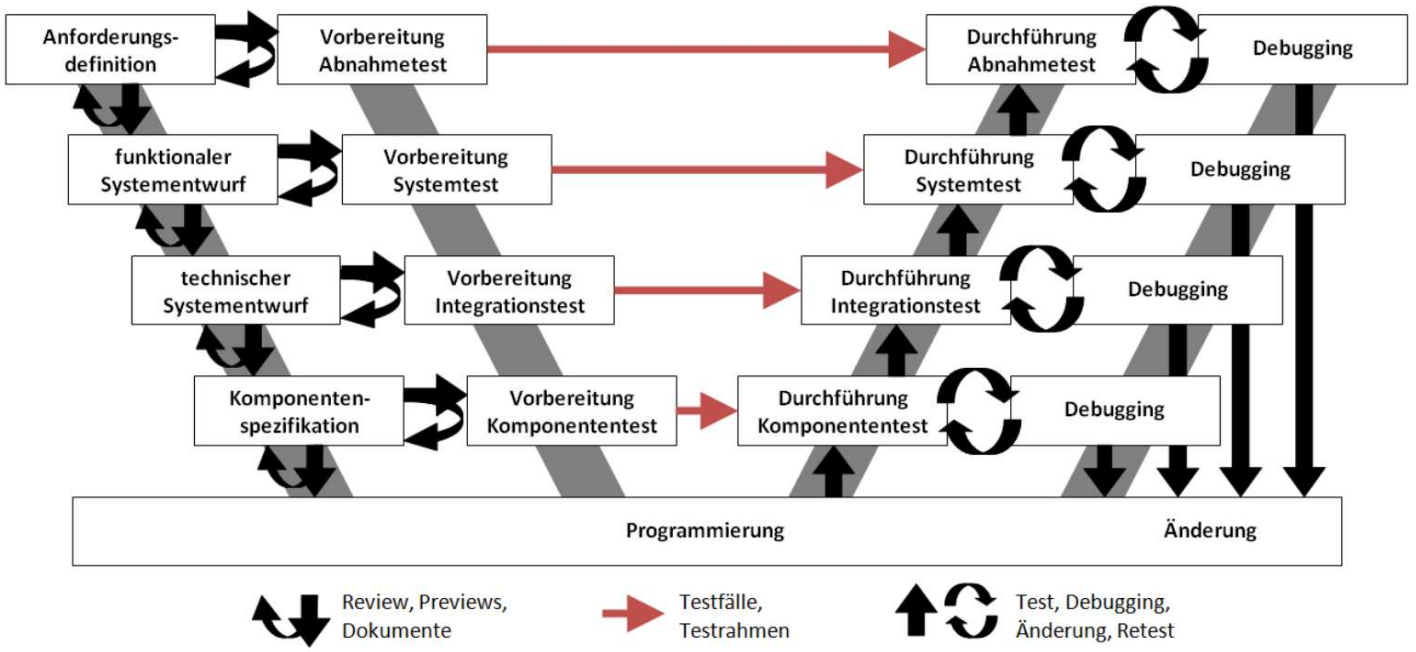
Black-Box/ White-Box



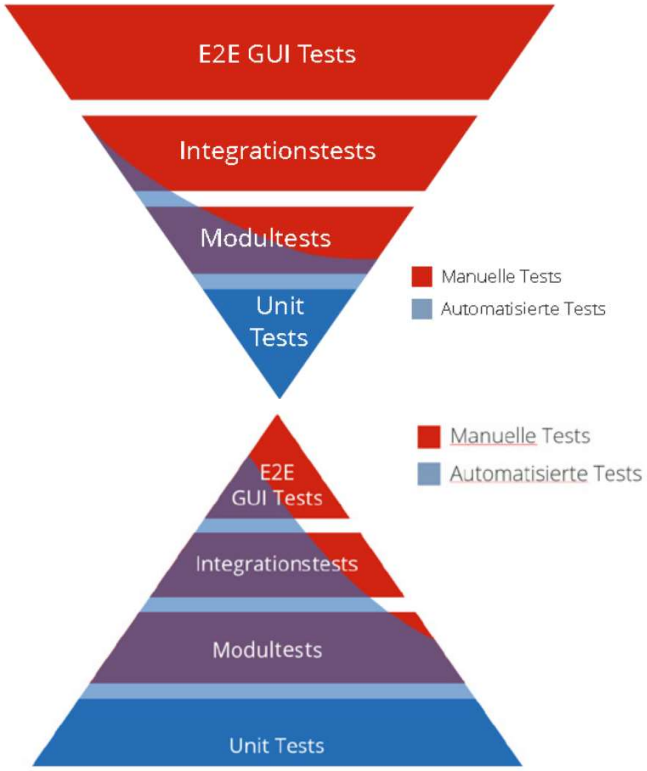
Teststufen - V-Modell



Teststufen - W-Modell



Testaufwand - Vergleich



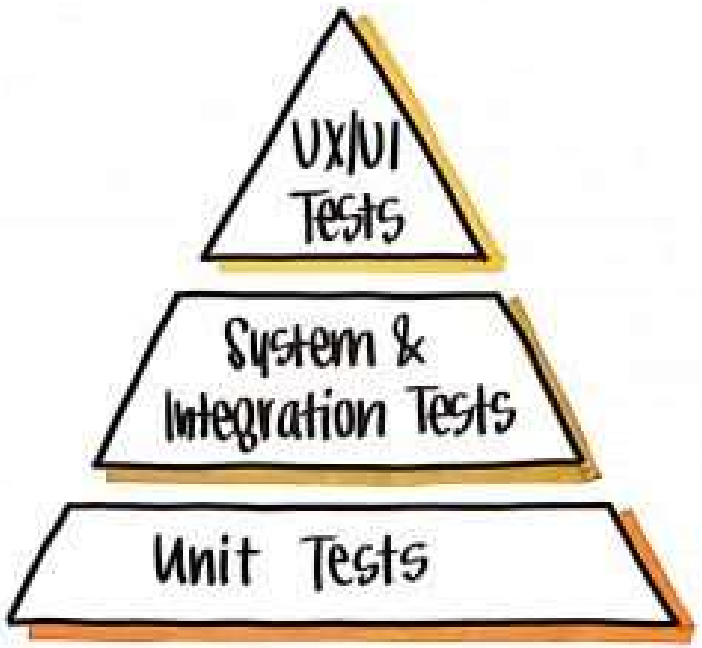
	durchschn. Anteil entdeckter Fehler	Kosten pro entdeckten Fehler	Total
E2E-GUI-Tests	35 %	20	7
Integrationstests	20 %	5	1
Modultests	15 %	3	0,5
Unit-Tests	30 %	1	0,3
	100 %		**8,8**

	durchschn. Anteil entdeckter Fehler	Kosten pro entdeckten Fehler	Total
E2E-GUI-Tests	5 %	10	0,5
Integrationstests	20 %	3	0,6
Modultests	30 %	2	0,6
Unit-Tests	45 %	1	0,5
	100 %		**2,2 statt 8,8**

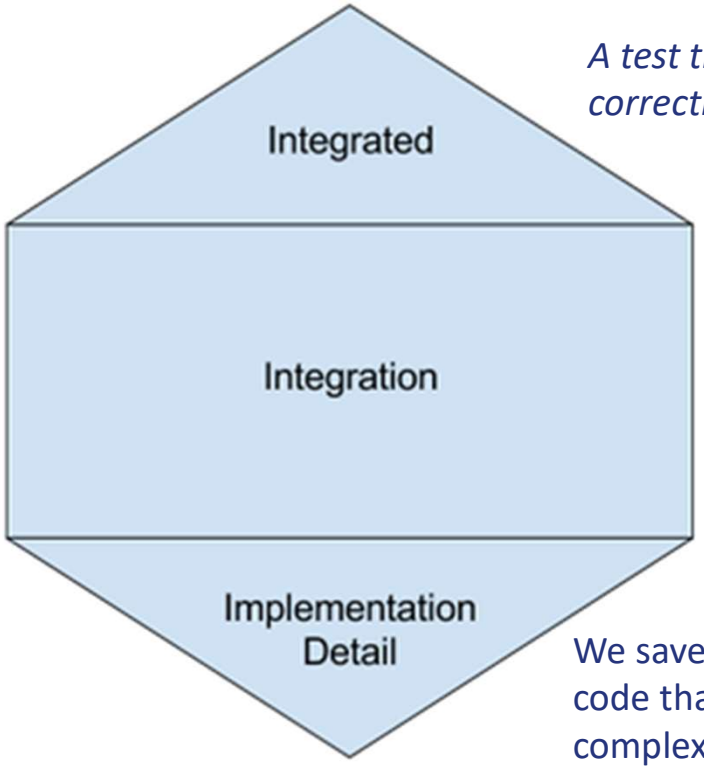
Monolith - Testpyramide

teuer, langsam

günstig, schnell



Microservices-testing-honeycomb (Spotify)



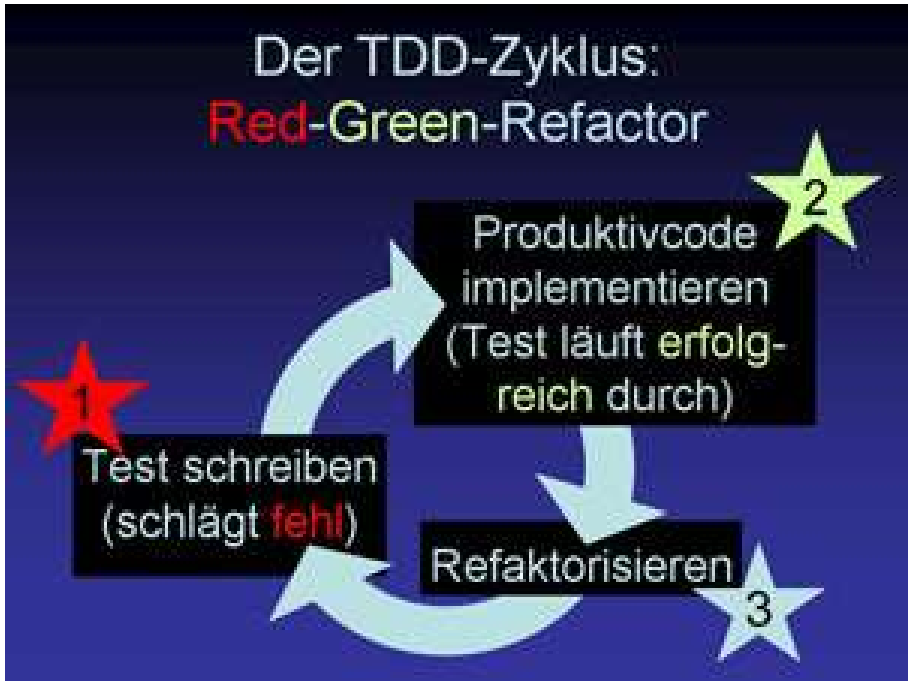
A test that will pass or fail based on the correctness of another system.

What we should aim for instead is Integration Tests, which verify the correctness of our service in a more isolated fashion while focusing on the interaction points and making them very explicit.

We save Implementation Detail Tests for parts of the code that are naturally isolated and have an internal complexity of their own.

https://engineering.atspotify.com/2018/01/testing-of-microservices/

Test Driven Development



Agenda

- Vorstellung
- Grundlagen
- **JUnit Framework**
- Mock-Objekte mit Mockito
- Fortgeschrittene Möglichkeiten
- Ausblick

JUnit Framework

- JUnit Architektur
- JUnit einbinden und verwenden
- Aufbau von einem Junit-Test
- Verwendung von Standard Assertions zur Prüfung
- Sammeln von Testfällen in Suits
- Tests in Testkategorien untergliedern
- Test-Reihenfolge
- Testen von Exceptions, Fehlerfälle
- Testen mit Timeouts
- Parametrisierte Tests
- Wiederholende Tests
- Verwendung von jAssert Assertions zur Prüfung

Vorab zum Code

- Der Code wird Euch über github zur Verfügung gestellt: https://github.com/MichaelZett
- Viele Dinge werde ich am Code zeigen. Da ist es sinnvoll vor allem zu zu gucken
- Es wird Aufgaben/Zeit geben, da könnt ihr dann selber arbeiten
- Ich entschuldige mich schon mal vorab über „Denglisch“ in Code und Folien

Grundlagen Java

- Entstanden 1995, aktuelle Version 19 (gerade erschienen)
- Seit Jahren immer eine der angesagtesten Programmiersprachen in den Umfragen
- Objekt-orientiert, seit Version 8 gibt es „funktionalen Zucker“
- Seit Java 9 erscheinen halbjährlich neue Versionen, alle paar Versionen gibt es Long Time Supported (LTS) Versionen, aktuelle LTS: Java 17
- OracleJDK für Business von Java 11 bis 16 kostenpflichtig (OpenJdk als Alternative), ab 17 LTS wieder frei auch für business

Grundlagen Eclipse

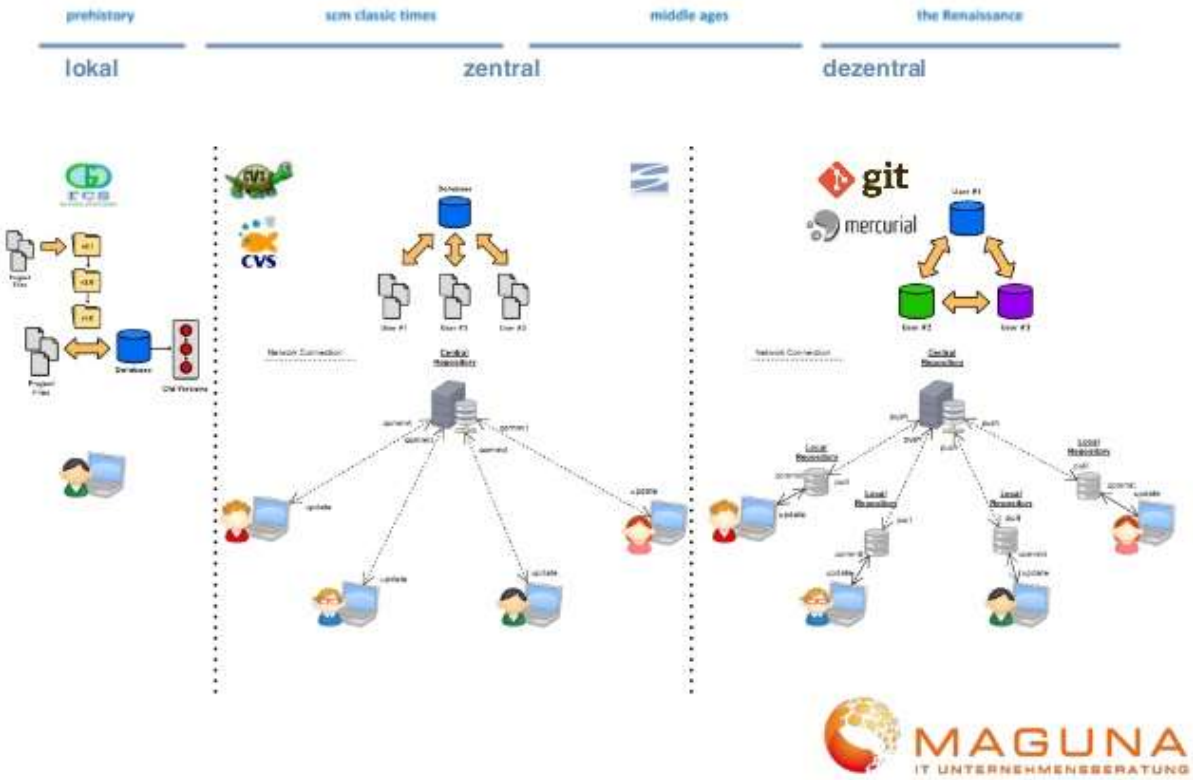
- Seit 2001 freie Integrated Development Environment
- Für Java aber auch alle möglichen anderen Sprachen
- Wird durch Plug-Ins aufgebaut
- Bis Photon (2018) erschien jedes Jahr im Juni eine neue Major-Version, danach ~3 Service Releases
- Seit 2018-09 nun 3-monatliche „RollingReleases“

Grundlagen IntelliJ

- Release 1.0 im Jahr 2001
- Hersteller: JetBrains (Entwickler der Sprache kotlin)
- Freie Community Version
- Kostenpflichtige Ultimate Edition
- Jährliches Major Release mit regelmäßigen Updates
- Durch Plug-Ins erweiterbar

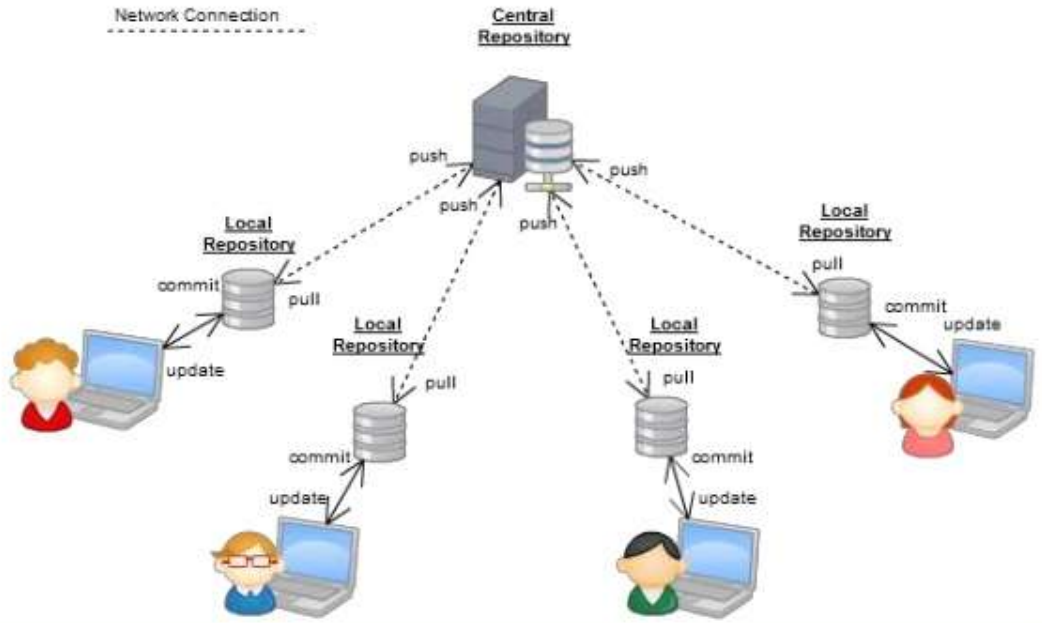
Exkurs Git

Geschichte von Versionsverwaltungen



MAGUNA
IT UNTERNEHMENSBERATUNG

Verteilte Versionsverwaltung



Git

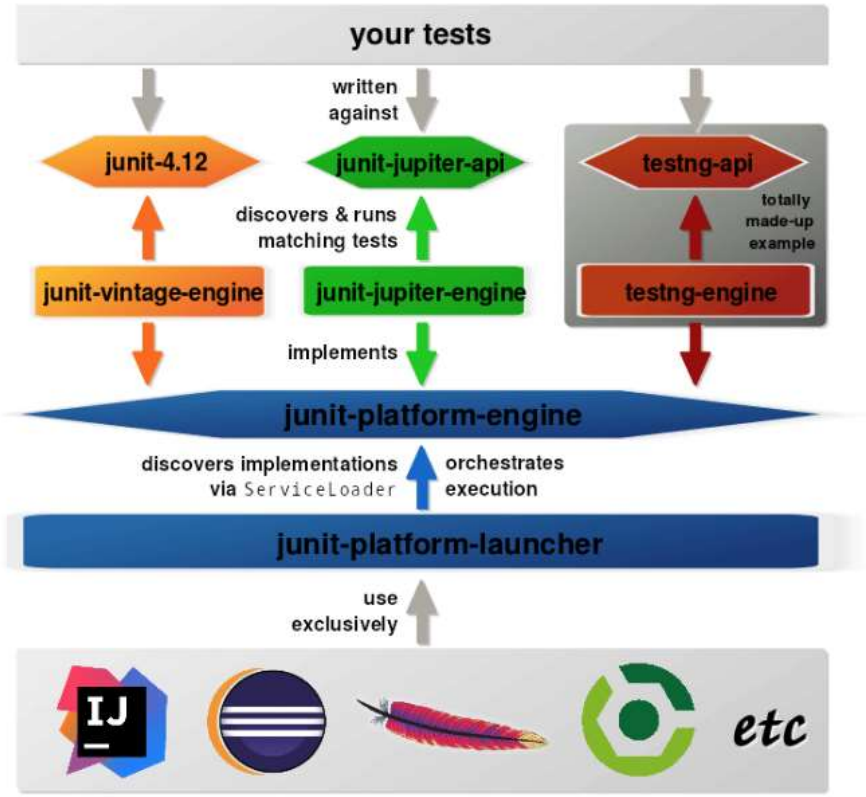
- Vom „Macher“ von Linux: Linus Torvalds
- Jeder hat das komplette Repo samt History lokal verfügbar
- Sehr effiziente Speicherung der Daten
- Branch und Merge integraler Bestandteil des Werkzeugs
- Wichtigste Befehle:
  - Clone: (Remote-)Projekt auschecken
  - Status: Unterschied zwischen lokal und remote feststellen
  - Fetch: Änderungen von remote holen ohne Verarbeitung
  - Pull/Rebase: Aktualisieren des lokalen repos von remote
  - Add: markieren von Änderungen für das commit
  - Commit: Einchecken ins lokale repo
  - Push: Hochladen ins remote repo

Unsere erste Klasse

- Einfaches IntelliJ Projekt öffnen

- Implementiere Klasse YearHelper mit einer Methode, die für eine Jahreszahl beantwortet, ob es sich um ein Schaltjahr handelt:
  - static boolean isSchaltjahr(int jahr)
  - Ein Jahr ist ein Schaltjahr
    - Wenn man es ganzzahlig durch 4 teilen kann und
    - Wenn man es nicht ganzzahlig durch 100 teilen kann, es sei denn es ist auch durch 400 ganzzahlig teilbar

JUnit Architecture



Unser erster Test

- Schreibe Tests zu der Klasse "YearHelper" mit einer Methode "boolean isSchaltjahr(int jahr)„
- Run test with coverage to check whether all relevant lines and branches have been checked
- **Run/EditConfiguration then proceed to Modify options/coverage setting and enable use tracing**

Aufbau

- Eine Test-Klasse pro zu testende Klasse
  - Namenskonvention:
    - Klasse "KlasseUnterTest"Test
    - KlasseUnterTest: testee
- Pro Test eine Methode mit @Test
- Testaufbau: Given-When-Then
  - Given: Vorbereitung der Testsituation (Annahmen, Daten, Mocks)
  - When: Aufruf der zu testenden Methode
  - Then: Prüfungen

Fizz Buzz Beispiel – Tests machen den Code besser

Schreibe ein Programm, das für die Zahlen von 1 bis 100 eine Ausgabe erstellt. Bei jeder Zahl, die durch 3 teilbar ist, soll "fizz" ausgegeben werden und bei jeder Zahl, die durch 5 teilbar ist, soll "buzz" ausgegeben werden. Wenn die Zahl sowohl durch 3 als auch durch 5 teilbar ist, soll "fizzbuzz" ausgegeben werden. Andernfalls wird die Zahl selbst ausgegeben.

Das Ergebnis soll in einer Komma-separierten Zeile stehen. Nach dem letzten Element soll kein Komma stehen.

Bisher benutzte Annotations

- *@Test - define Test method*
- *@DisplayName* – defines custom display name for a test class or a test method
- *@BeforeEach* - denotes that the annotated method will be executed before each test method (previously in Junit 4 *@Before*)
- *@AfterEach* – denotes that the annotated method will be executed after each test method (previously in Junit 4 *@After*)
- *@BeforeAll* – denotes that the annotated method will be executed before all test methods in the current class (previously *@BeforeClass*)
- *@AfterAll* – denotes that the annotated method will be executed after all test methods in the current class (previously *@AfterClass*)
- *@Disable* – it is used to disable a test class or method (previously *@Ignore*)
- *@TestInstance(TestInstance.Lifecycle.PER_METHOD) - default - new test instance before each method (test)*

Bisher benutzte Assertions

- *assertTrue – method or statement returns boolean true*
- *assertFalse – method or statement returns boolean false*
- *assertEquals – expected and actual are equal*

Assertions

- https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-assertions-with-junit-5-api/
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions
- Unterordner: assertions

Weitere Assertions und Annotations

- *assertIterableEquals – 2 Iterables are equal*
- *assertArrayEquals – 2 Arrays are equal*
- *assertNull – reference is null*
- *assertNotNull – reference is not null*
- *assertSame - reference is identical*
- *assertNotSame - reference is not identical*

- *@Nested* - denotes that the annotated class is a nested, non-static test class (sub structure)

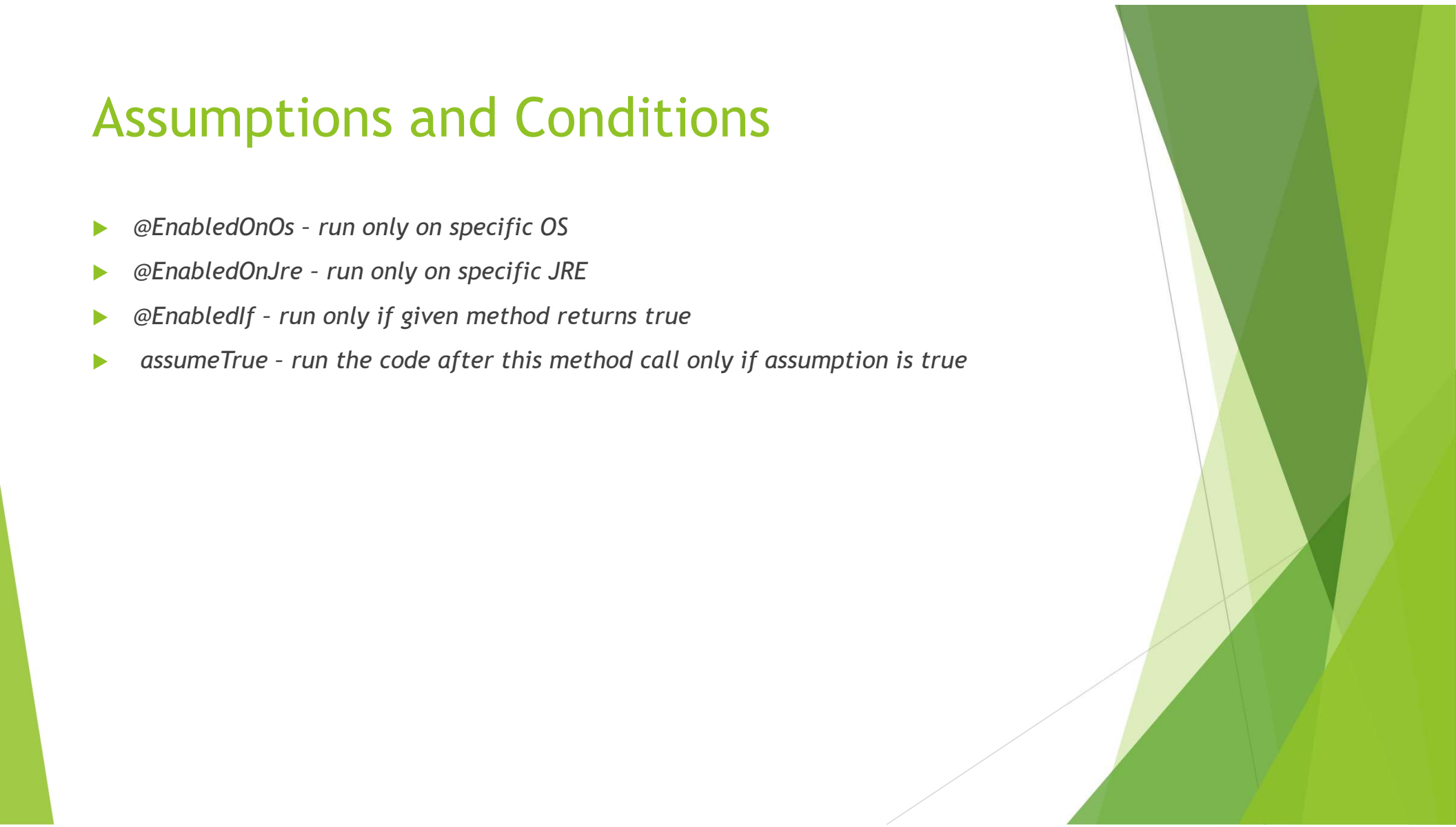
Assumptions and Conditions

- https://junit.org/junit5/docs/current/user-guide/#writing-tests-assumptions
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-disabling
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-conditional-execution

- Unterordner: assumptions

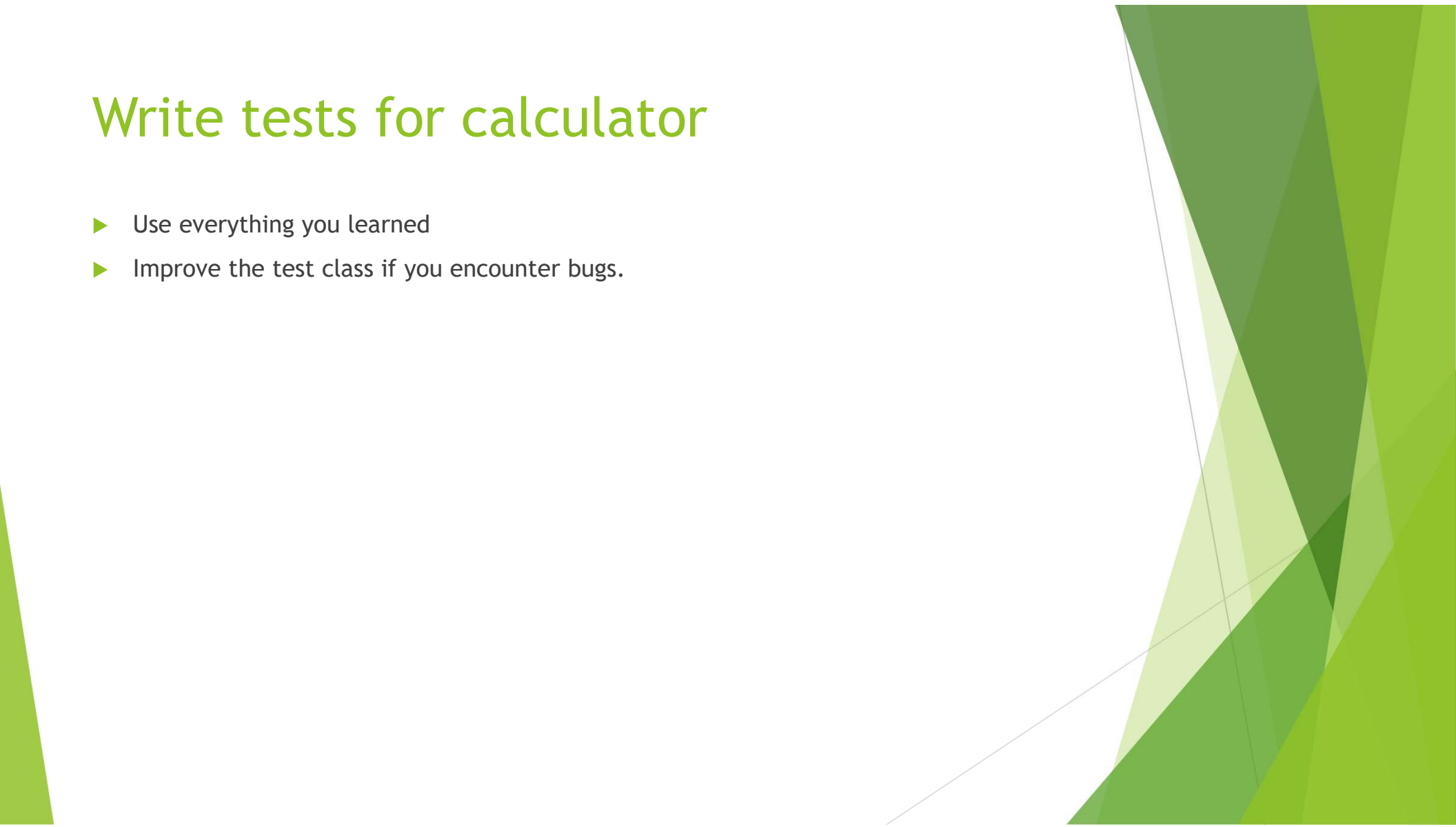
Assumptions and Conditions

- *@EnabledOnOs – run only on specific OS*
- *@EnabledOnJre – run only on specific JRE*
- *@EnabledIf – run only if given method returns true*
- *assumeTrue – run the code after this method call only if assumption is true*



Write tests for calculator

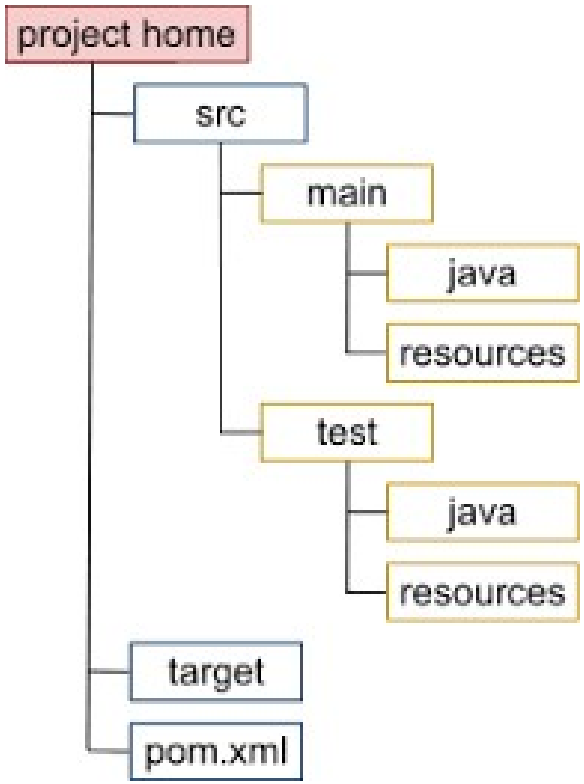
- Use everything you learned
- Improve the test class if you encounter bugs.



Exkurs Maven

- Seit 2001
- Build-System
- Dependency-Management
- Alternative: gradle

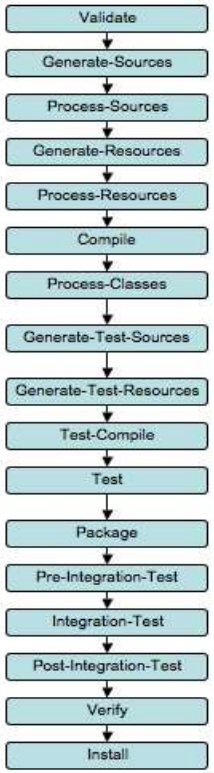
Folders


project home
src
main
java
resources
test
java
resources
target
pom.xml

Maven Lifecycle

Clean (eigener Befehl)

Validate
Compile
Test
Package
Verify
Install
Deploy



Grouping, Test Suites, Reihenfolge

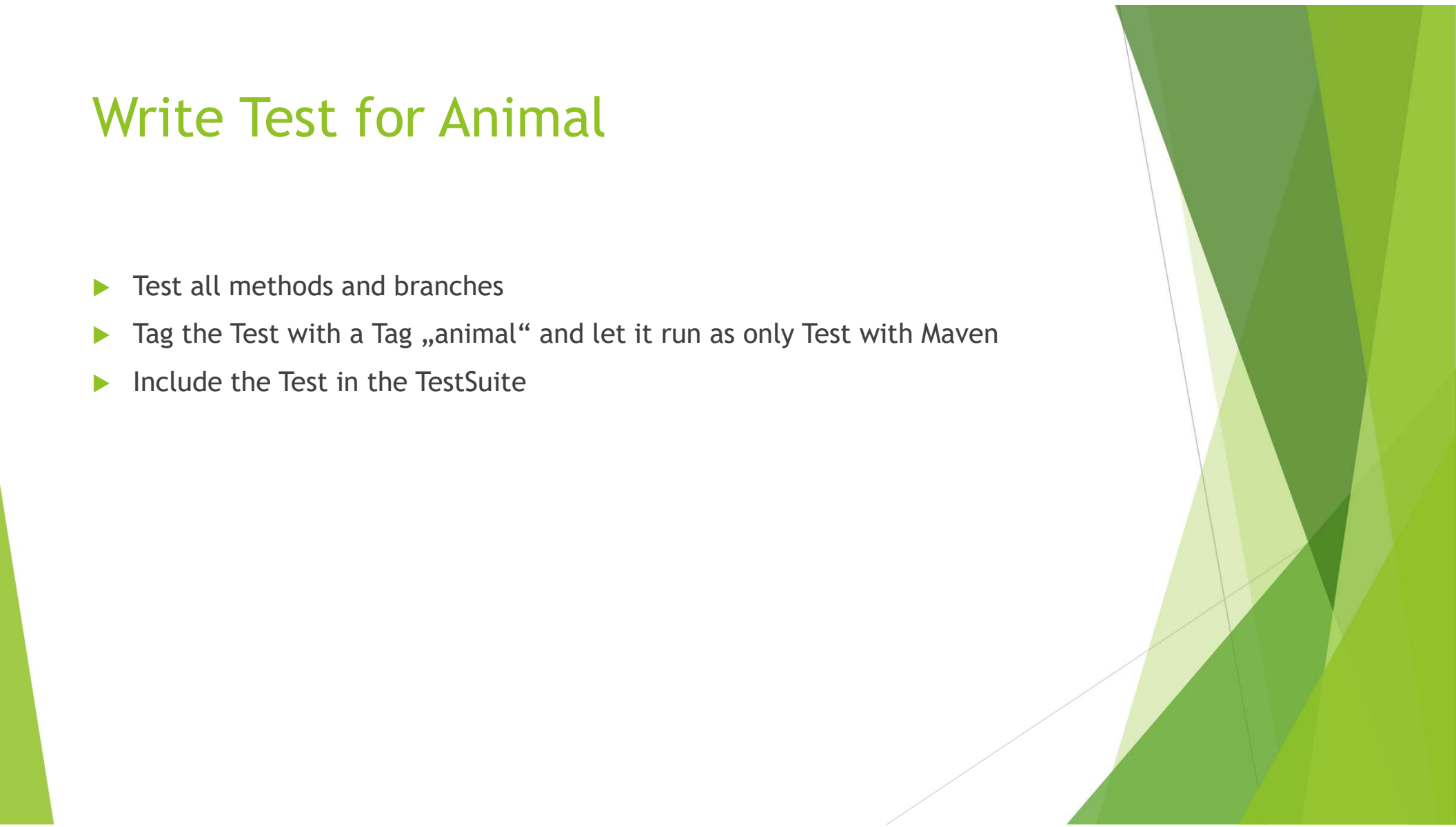
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-tagging-and-filtering
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-execution-order
- https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-assertions-with-junit-5-api/
- Grouping assertions
- Legacy Junit 4 mit Junit 5 verwenden
- Projekt: grouptagfilter

What did we learn?

- *assertAll – all included assertions are run and reported together (does not stop at the first assertion failing)*
- *Some old JUnit 4 stuff (@Test,@Before)*
- *@Suite, @SuiteDisplayName("This is a suite") – definition of a Suite and its name*
- *@SelectPackages, @SelectClasses – what should be run*
- *@Tag, @ExcludeTags – what should or should not be run*
- *Using maven to run only special groups of tests*
- *@TestMethodOrder, @Order – define ordering of the tests*

Write Test for Animal

- Test all methods and branches
- Tag the Test with a Tag „animal“ and let it run as only Test with Maven
- Include the Test in the TestSuite



Exceptions, Timeouts, Repeated, Parametrisierte Tests

- https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-assertions-with-junit-5-api/
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-declarative-timeouts
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-repeated-tests
- https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests

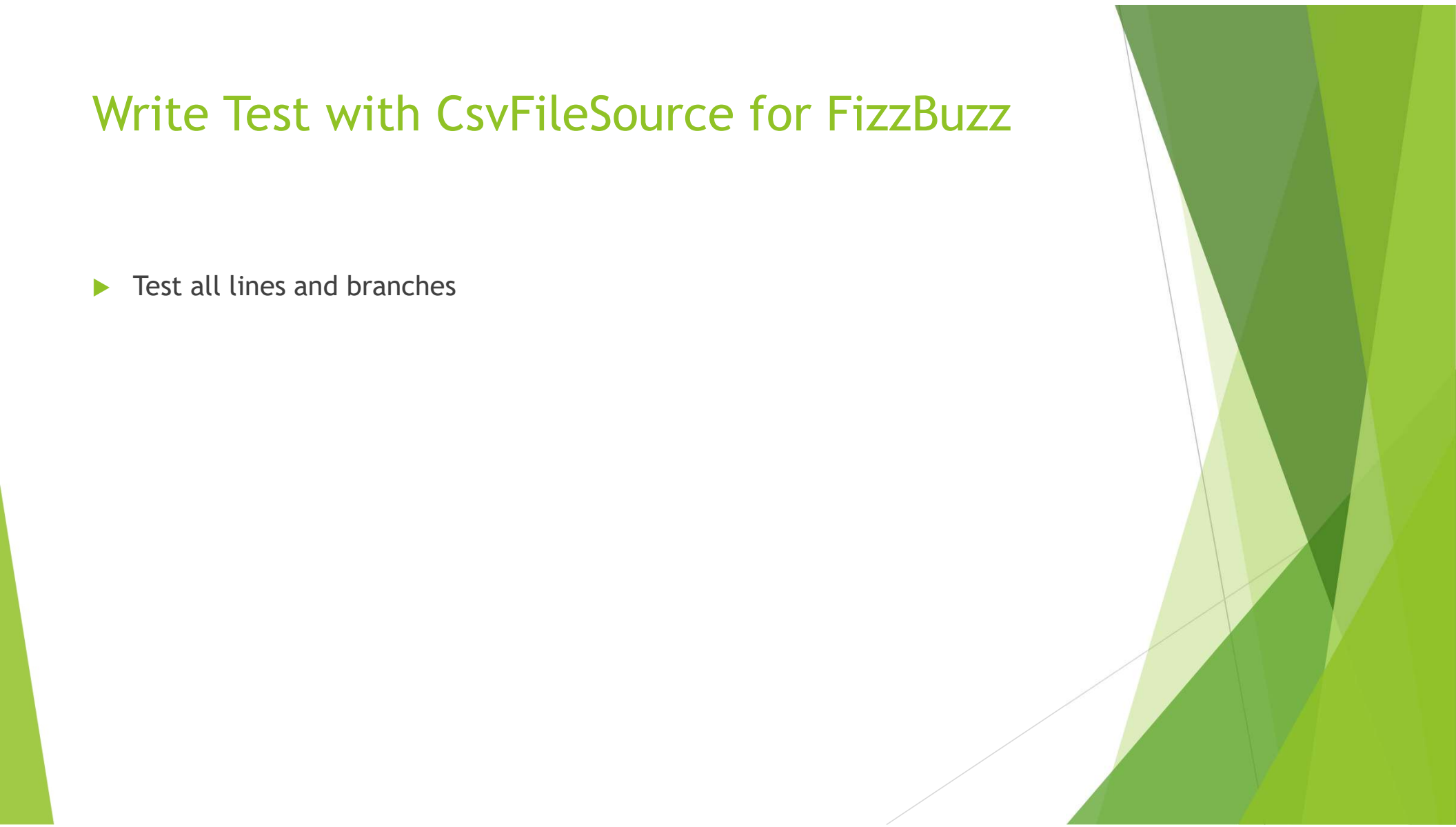
- Projekt: exception_time_param

What did we learn?

- *assertThrows – expect exeptions*
- *assertTimeout – expect timeout*
- *Legacy JUnit4-Rule – mechanism to extend JUnit4*
  - *ExpectedException - expect exeptions (but even with Junit 4 we can get rid of it)*
- *@ParameterizedTest – needed for parameterized Tests*
  - *@ValueSource – Give values directly*
  - *@MethodSource – use a method for values*
  - *@CsvSource – give values in CSV style directly*
  - *@CsvFileSource – use a .csv file for values*
- *@RepeatedTest – run a test a repeated time*

Write Test with CsvFileSource for FizzBuzz

- Test all lines and branches

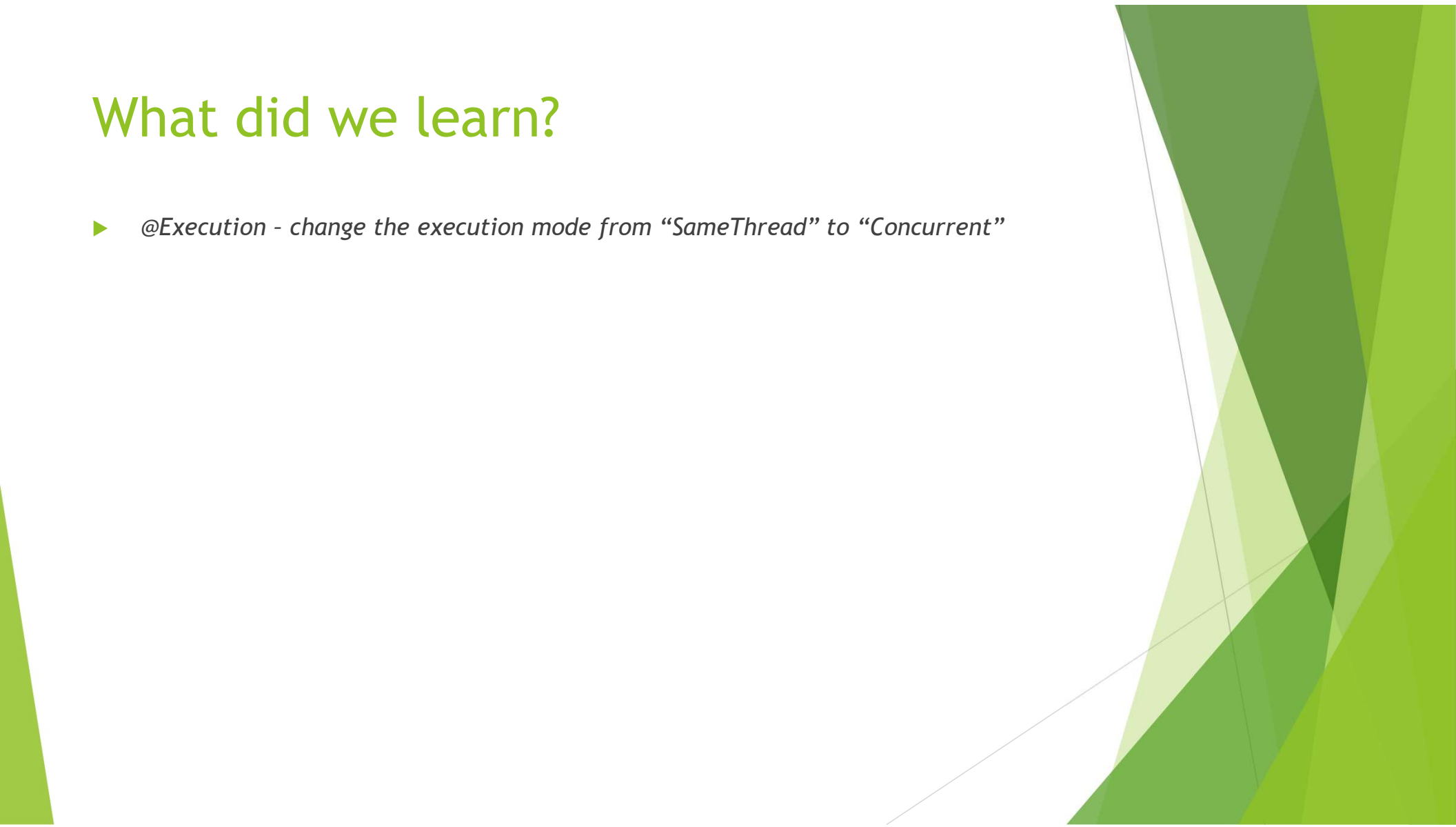


Experimental: parallel execution

- https://junit.org/junit5/docs/current/user-guide/#writing-tests-parallel-execution
- Projekt: parallel
- Running parallel with maven: https://www.baeldung.com/maven-junit-parallel-tests

What did we learn?

- *@Execution – change the execution mode from “SameThread” to “Concurrent”*



Test coverage in builds

- Jacoco: https://www.eclemma.org/jacoco/trunk/doc/maven.html
- clean verify jacoco:report

Hamcrest, AssertJ et al

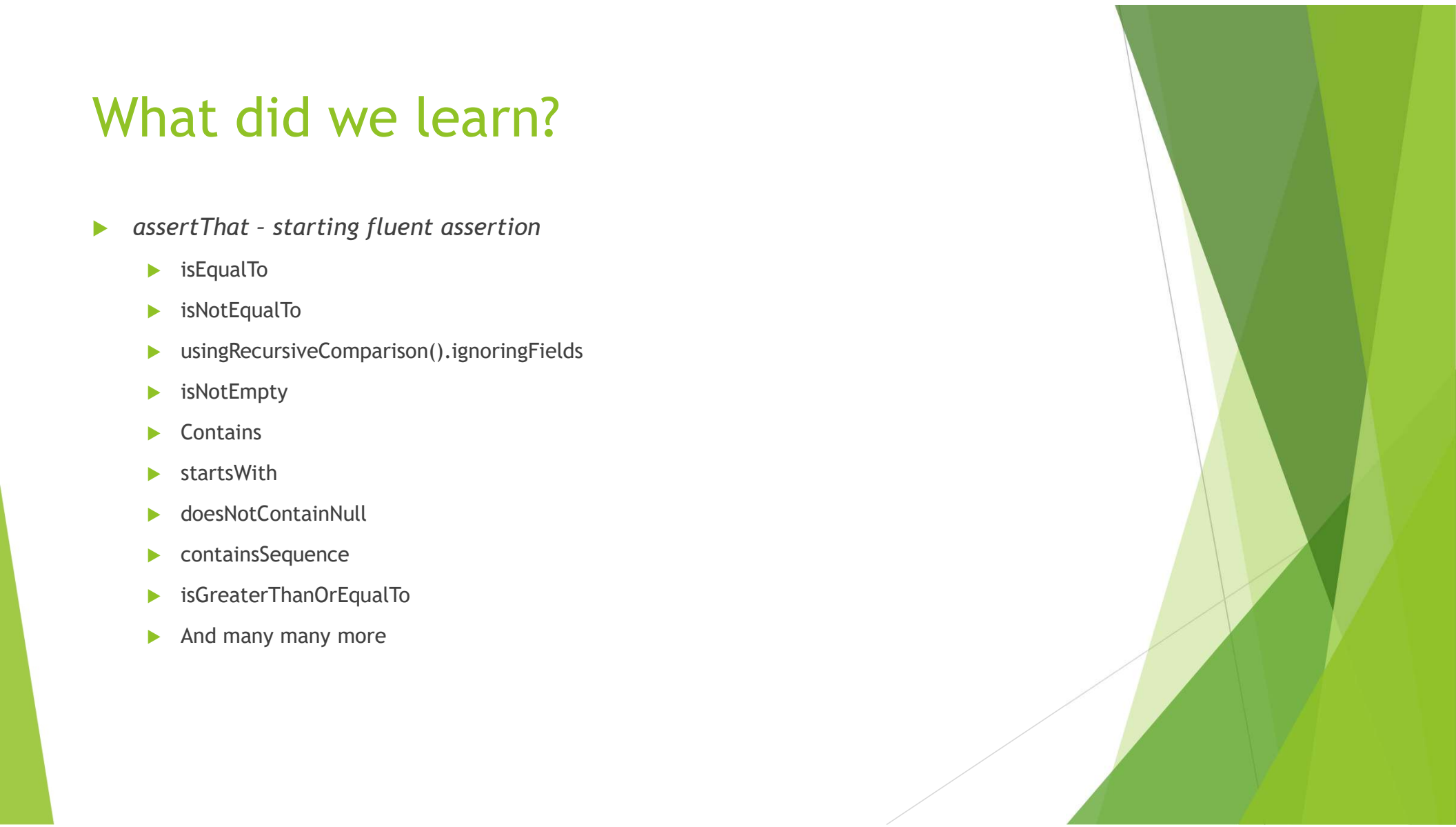
- http://hamcrest.org/JavaHamcrest/index
- https://assertj.github.io/doc/
- https://dzone.com/articles/hamcrest-vs-assertj-assertion-frameworks-which-one
- Projekt: assertjdemo

Sonarqube - test coverage and more

- https://www.sonarqube.org/
- mvn clean package sonar:sonar

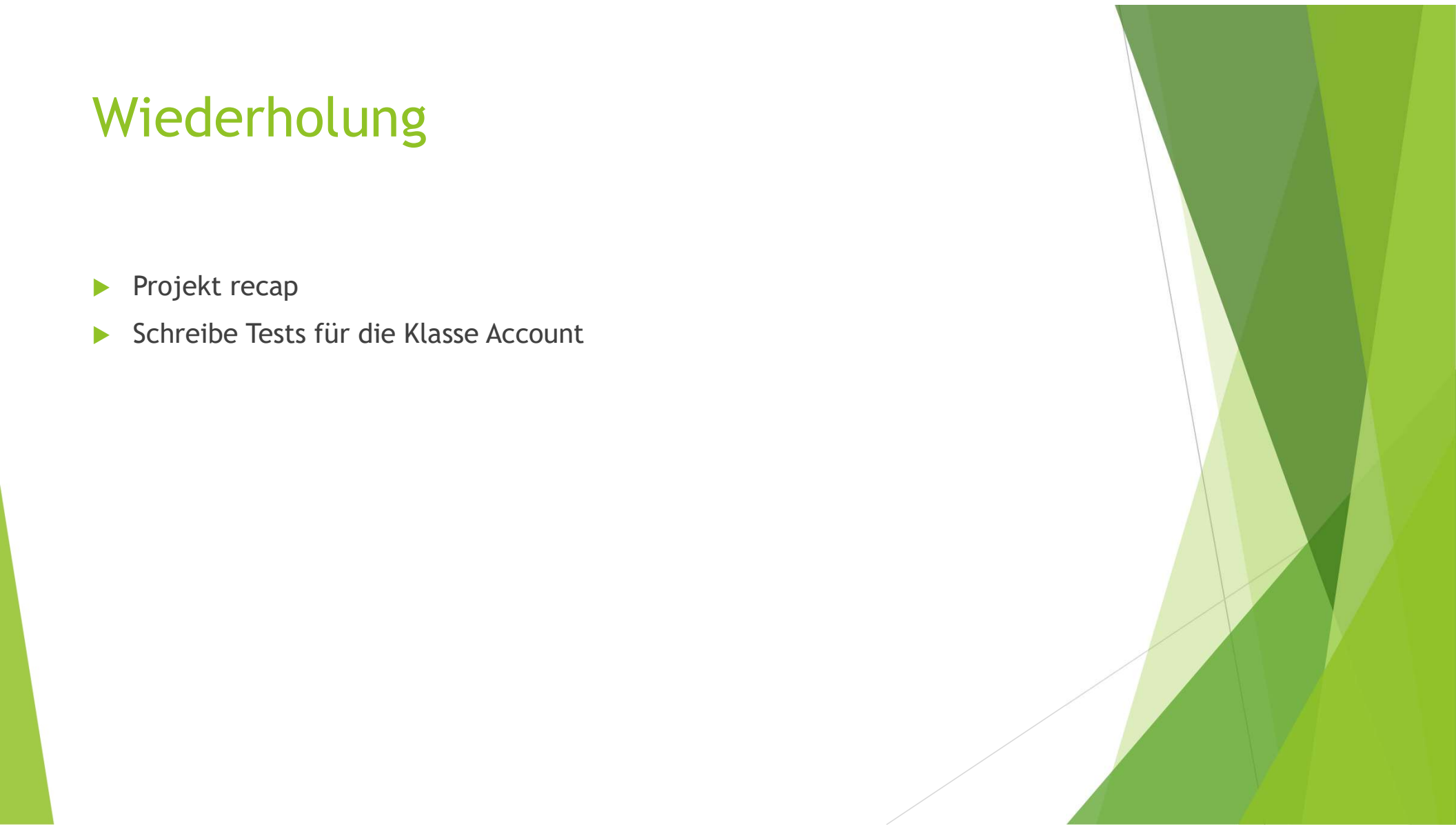
What did we learn?

- *assertThat – starting fluent assertion*
  - isEqualTo
  - isNotEqualTo
  - usingRecursiveComparison().ignoringFields
  - isNotEmpty
  - Contains
  - startsWith
  - doesNotContainNull
  - containsSequence
  - isGreaterThanOrEqualTo
  - And many many more



Wiederholung

- Projekt recap
- Schreibe Tests für die Klasse Account



What did we learn?

- *@TestInstance(TestInstance.Lifecycle.PER_CLASS) – use class and non-static testee for all tests*

Mock-Objekte mit Mockito

- Erzeugen von Mock-Objekte
- Anzahl von Methodenaufrufe prüfen
- Exceptions von Mock-Objekten auslösen
- Callbacks

https://site.mockito.org/

Unterschiede von Test-Objekten

- A *dummy object* is passed around but never used, i.e., its methods are never called. Such an object can for example be used to fill the parameter list of a method.
- *Fake* objects have working implementations, but are usually simplified. For example, they use an in memory database and not a real database.
- A *stub* class is an partial implementation for an interface or class with the purpose of using an instance of this stub class during testing. Stubs usually don't respond to anything outside what's programmed in for the test. Stubs may also record information about calls.
- A *mock object* is a dummy implementation for an interface or a class in which you define the output of certain method calls. Mock objects are configured to perform a certain behavior during a test. They typically record the interaction with the system and tests can validate that.

Mockito – Grundlegende Elemente

- Verhalten spezifizieren
  - when(mock.method()).thenReturn(result)
- Aufrufe erwarten
  - verify(mock, times(2)).method();
- Exceptions spezifizieren
  - doThrow(Exception.class).when(mock).method();

- Projekt: mocks

What did we learn?

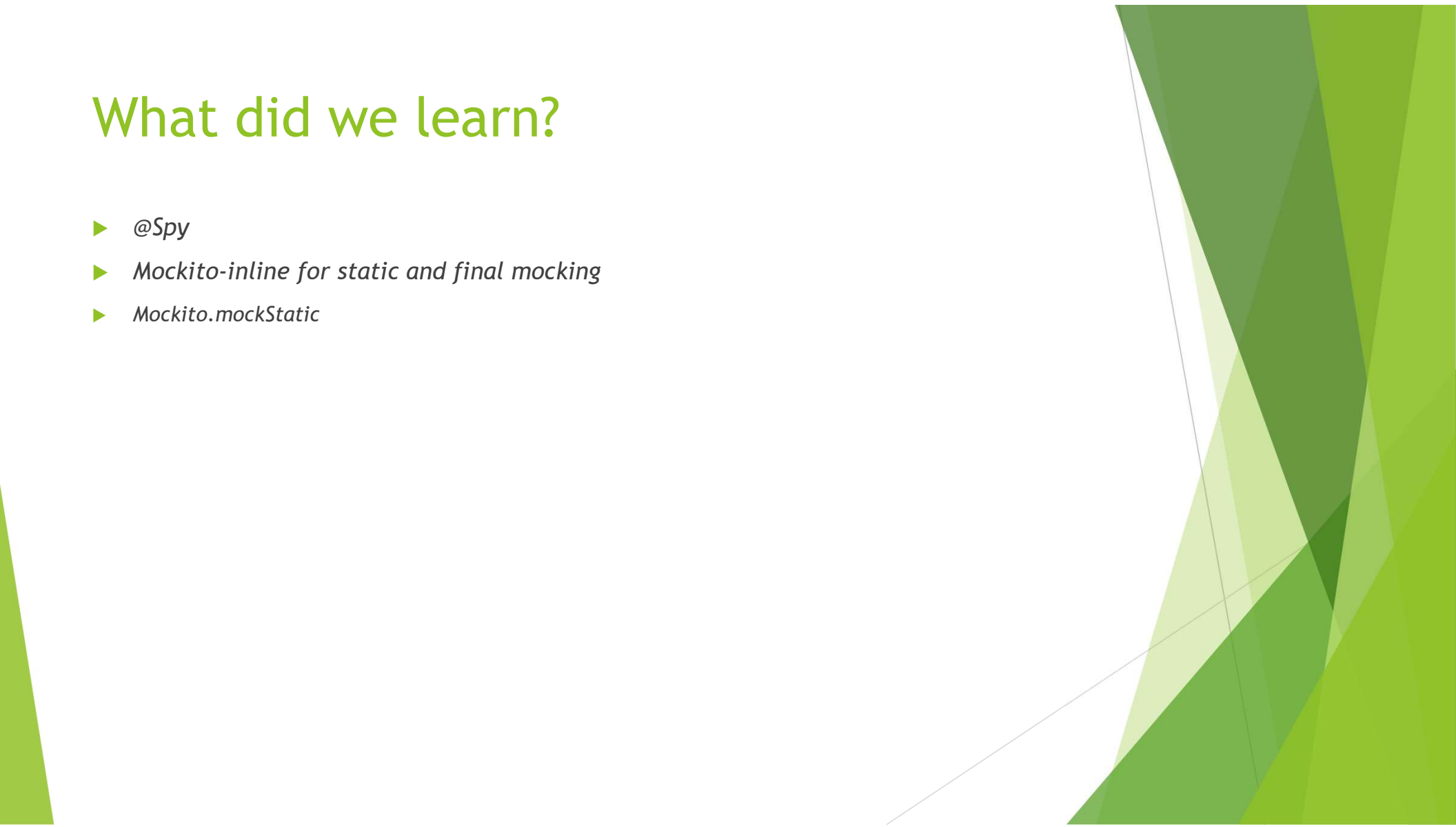
- *@ExtendWith – use extensions for Junit 5*
  - *@Mock – create a mock from the type*
  - *@InjectMocks – Add created mocks automatically into the testee*
  - *@Captor – create an argument captor*
- *@MockitoSettings – change settings like strictness*
- *mock() – create a mock from the type*
- *verify – check that a specific call has happended*
  - *ArgumentMatcher – check parameters used in call*
    - *Any – any value of a given class*
    - *Eq – needs to equal a specific value*
  - *InOrder – check order of calls*
  - *verifyNoMoreInteractions – only the previously checked calls*

Mockito – Fortgeschrittene Techniken

- Stubs erstellen
  - Spy
- Callbacks
  - when(calcService.add(20.0,10.0)).thenAnswer(new Answer<Double>() {
  - @Override
  - public Double answer(InvocationOnMock invocation) throws Throwable {
  - //get the arguments passed to mock
  - Object[] args = invocation.getArguments();
  - //get the mock
  - Object mock = invocation.getMock();
  - //return the result
  - return 30.0;
  - }
  - });

What did we learn?

- *@Spy*
- *Mockito-inline for static and final mocking*
- *Mockito.mockStatic*

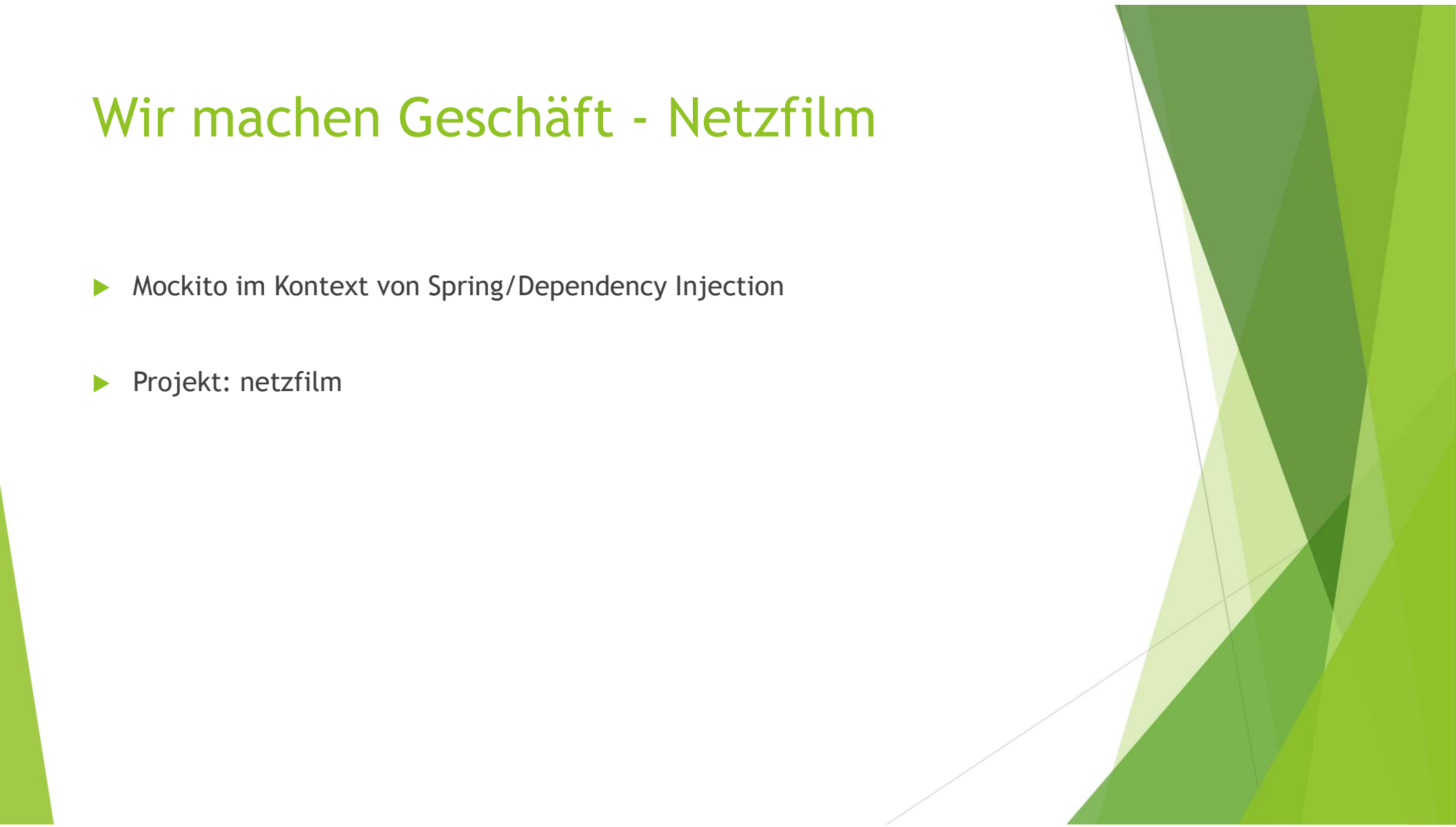


Repair the weather feature

- Der WeatherService erfüllt die Anforderungen!
  - Gutes Wetter nur wenn alle Bedingungen gut sind
  - Schlechtes Wetter wenn eine Bedingung schlecht ist, es sei denn der schlechte Bedingung steht auch eine gute gegenüber, dann ist das Wetter ok
  - Sonst OK
- Schreibe alle Tests
- Findest Du einen Fehler, behebe ihn

Wir machen Geschäft - Netzfilm

- Mockito im Kontext von Spring/Dependency Injection
- Projekt: netzfilm



What did we learn?

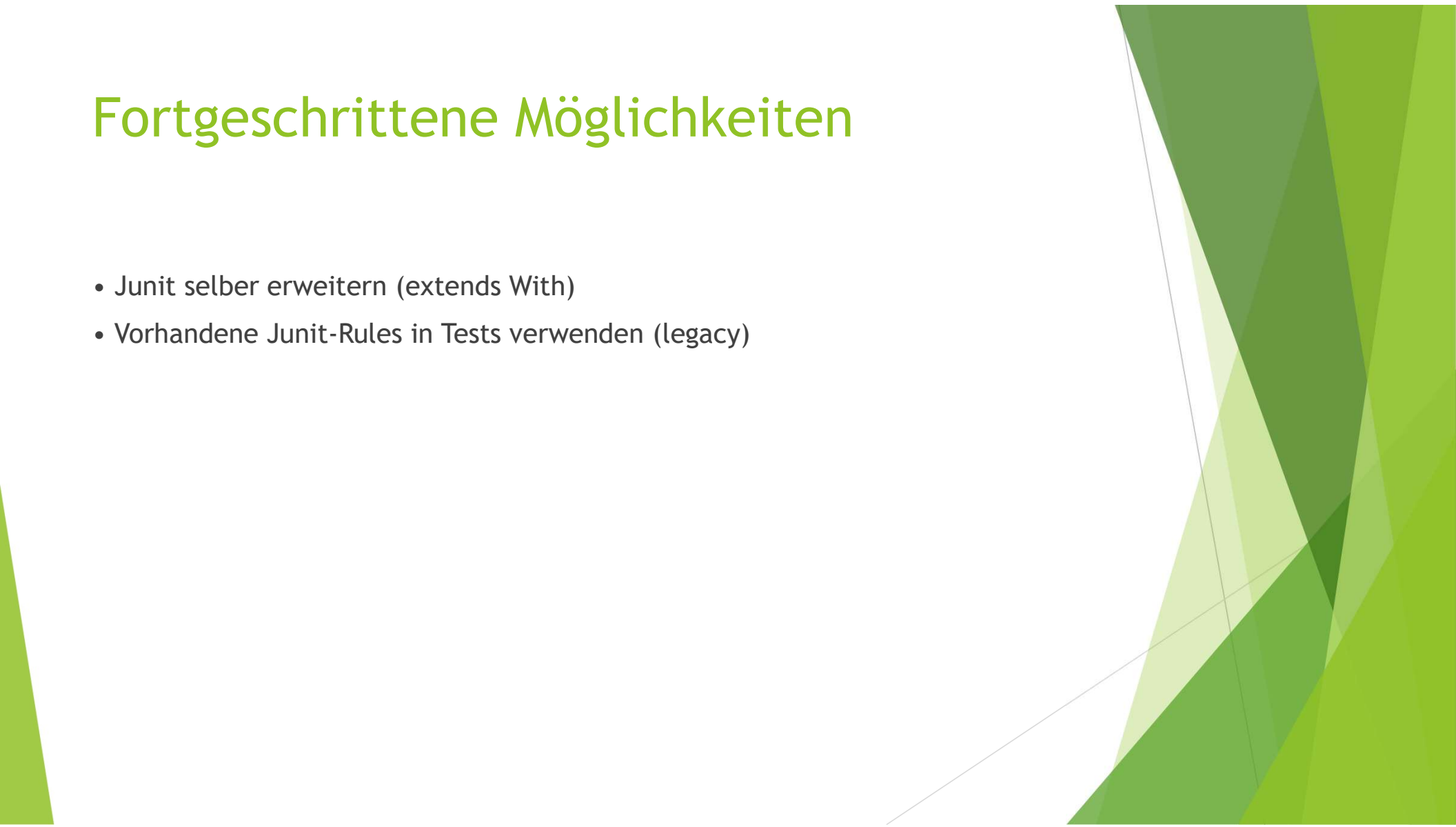
- *@SpringBootTest - create a Spring Boot Test (app is started for test)*
- *@WebMvcTest - test for controllers*
  - *Perform*
  - *andExpect*
- *@DataJpaTest - test for repositories*
- *@TestPropertySource - add properties to test execution*

Customer testen

- CustomerRepositoryTest (DataJpaTest)
- CustomerServiceImplTest (MockitoTest)
- CustomerControllerWebMvcTest (WebMvcTest)

Fortgeschrittene Möglichkeiten

- Junit selber erweitern (extends With)
- Vorhandene Junit-Rules in Tests verwenden (legacy)



Junit-4-Rules

```
public class HasTempFolderTest {

    @Rule
    public TemporaryFolder folder= new TemporaryFolder();

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

}
```

Junit-5-Extension

JUnit 5 extensions are related to a certain event in the execution of a test, referred to as an extension point. When a certain life cycle phase is reached, the JUnit engine calls registered extensions.

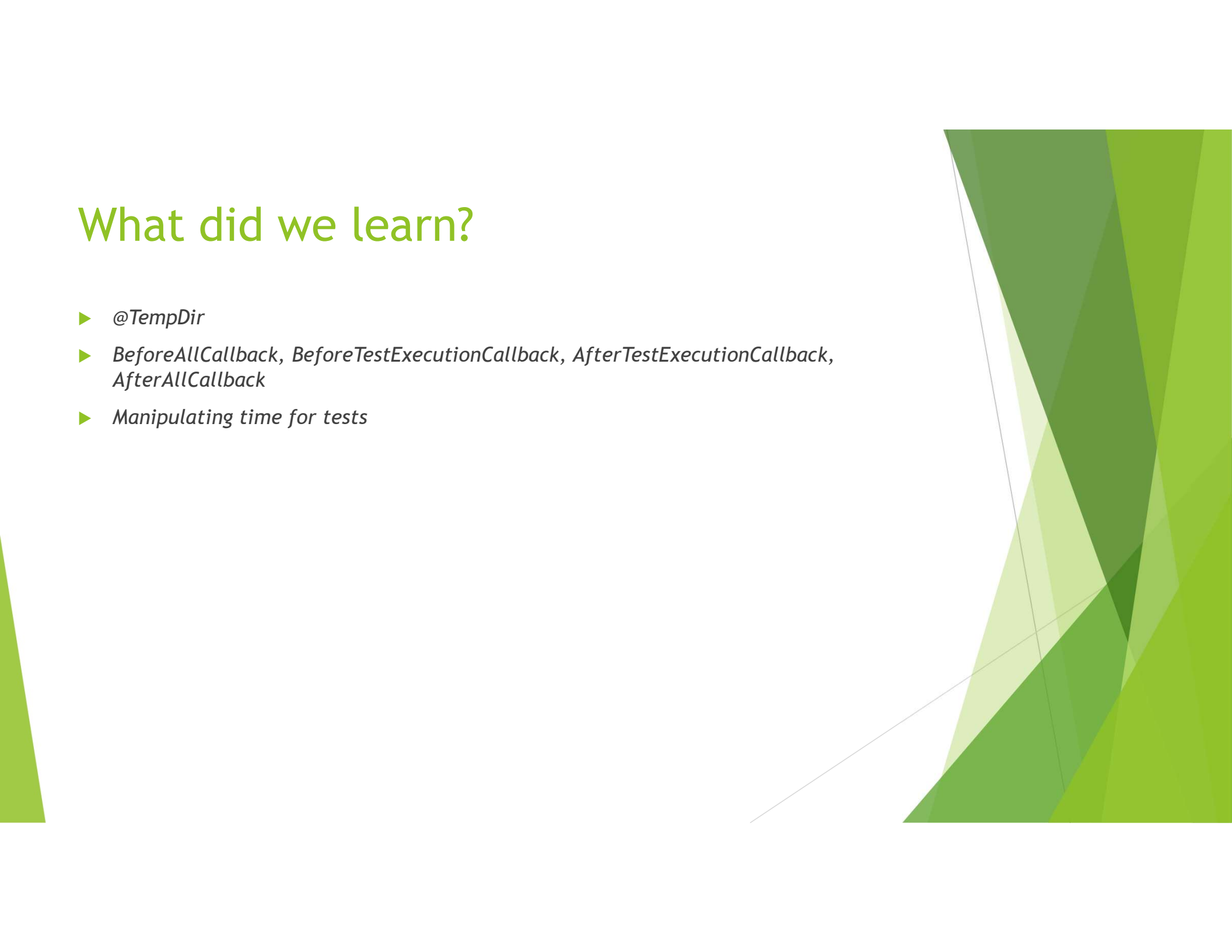
Five main types of extension points can be used:

- test instance post-processing
- conditional test execution
- life-cycle callbacks
- parameter resolution
- exception handling

- Projekt: extension

What did we learn?

- *@TempDir*
- *BeforeAllCallback, BeforeTestExecutionCallback, AfterTestExecutionCallback, AfterAllCallback*
- *Manipulating time for tests*



Testcontainer

- Alternative zu h2 – db per docker container
- Auch alle möglichen anderen Systeme nutzbar

- Projekt: 10-netzfilm

Ausblick

- Selenium – Tests durch Bedienung der Web-Oberfläche (E2E)
  - https://selenium.dev/
- WireMock – http-Schnittstellen mocken
  - https://wiremock.org/
- ArchUnit – Architekturregeln testen
  - https://github.com/TNG/ArchUnit
- BDD – Cucumber
  - https://cucumber.io/docs/bdd/