class: inverse, center, middle

SZT-01 - Szoftvertesztelés Java platformon

Java teszt eszközkészlet 1.

- JUnit
- Hamcrest
- AssertJ
- Mockito
- Jacoco
- XmlUnit
- DbUnit

Java teszt eszközkészlet 2.

- SoapUI
- Arquillian
- Concordion
- JMeter
- Selenium
- SonarQube, SonarLint

---

class: inverse, center, middle

Bevezetés a JUnit használatába

Unit tesztelés

- Forráskód legkisebb egységének tesztelésére
- OO nyelv (pl. Java) esetén osztály
- Az adott egység helyesen oldja meg a rá bízott feladatot
- Előírt bemenetre az elvárt kimenetet adja-e
- Tipikusan automatizált
- Tipikusan a szoftverfejlesztők írják

Mi nem unit teszt

- Több osztály kapcsolata

- Keretrendszer bevonása, pl. Spring, Java EE
- Adatbázis műveletek bevonása

JUnit

- Java unit teszt keretrendszer
- Kent Beck (Agile Manifesto, Extreme programming), Erich Gamma (Programtervezési minták)
- Ant, Maven, Gradle, IDE támogatás
- JUnit 4, JUnit 5
- Java osztály - *test class*
- `@Test` annotációval ellátott metódus - *test case*
- Assert statikus metódusok

Tesztelendő kód

```
public class Employee {

  private String name;

  private int yearOfBirth;

  // Konstruktorok

  public int getAge(int atYear) {
    return atYear - yearOfBirth;
  }

  // Getter és setter metódusok

}
```

Given - when - then

- Tesztesetek felépítésének egy stílusa
- Behavior-Driven Development (BDD) részeként
- Given: előfeltételek
- When: tesztelendő funkció hívása
- Then: elvárt eredmény

Metódus tesztelés

```
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;

public class EmployeeTest {

    @Test
    void testGetAge() {
        // Given
        Employee employee = new Employee("John Doe", 1970);

        // When
        int age = employee.getAge(2019);

        // Then
        assertEquals(49, age);
    }
}
```

Metódus tesztelés egybevonva

```
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import static org.junit.jupiter.api.DynamicTest.dynamicTest;

public class EmployeeTest {

    @Test
    void testGetAge() {
        assertEquals(49, new Employee("John Doe", 1970).getAge(2019));
    }
}
```

Maven integráció

- Maven életciklus - *test* fázis
- Surefire plugin - *test* cél, `2.22.0` verziótól támogatja a JUnit 5-öt
- Tesztesetek az `src/test/java` könyvtárban, hozzá tartozó erőforrások a `src/test/resources` könyvtárban

---

pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
```

```
        <version>${junit5.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
        </plugin>
    </plugins>
</build>
```

Unit tesztelés ígéretei

- Hibák mielőbbi megtalálása, ezáltal költségcsökkentés
- Refactoring
- Hibajavításkor regresszió, funkció nem romolhat el, hiba nem jelenhet meg újra
- Komplex rendszer esetén hamarabb találjuk meg a hibás komponenst
- High cohesion, osztály használható interfésszel rendelkezik
- Komponensek közötti interakció és függőségek mielőbbi átgondolása
- Tiszta, lazán kapcsolt kód, low coupling
- Fejlesztés egyszerűsítése
- Dokumentációként viselkedik

---

class: inverse, center, middle

Tesztesetek életciklusa

Test fixture

- Minden, ami szükséges a teszteset sikeres lefutásához (inicializálás - ismert állapotba hozás, előfeltételek)
- Kiemelhető külön metódusba
- `@BeforeEach`, `@AfterEach` annotációkkal ellátott metódusok
- `@BeforeAll`, `@AfterAll` annotációkkal ellátott *statikus* metódusok

Test fixture példa

```
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
```

```java
public class EmployeeTest {

    Employee employee;

    @BeforeEach
    void initEmployee() {
        employee = new Employee("John Doe", 1970);
    }

    @Test
    void testGetAge() {
        assertEquals(49, employee.getAge(2019));
    }

    @Test
    void testWithZeroAge() {
        assertEquals(0, employee.getAge(1970));
    }
}
```

Izoláltság

- Törekedni az izoláltságra, a unit tesztek függetlenségére
- Izoláltságot megszegi az állapot átvitele teszt esetek között (pl. statikus attribútumok értékei)
- JUnit támogatás: minden teszteset futtatásakor újra példányosítja a tesztosztályt

---

Példányosítás

```java
    @BeforeEach
    void initEmployee() {
        employee = new Employee("John Doe", 1970); // INIT
    }

    @Test
    void testGetAge() {
        assertEquals(49, employee.getAge(2019)); // TC1
    }

    @Test
    void testWithZeroAge() {
        assertEquals(0, employee.getAge(1970)); // TC2
    }
```

Lefutási sorrend: példányosítás, INIT, TC1, példányosítás, INIT, TC2

Sorrend

- Nem magától érthetődő, de determinisztikus - azaz ismételhető
- Integrációs teszteknél szükség lehet a sorrend meghatározására
  - `@TestMethodOrder` annotációval, `MethodOrderer` implementációval paraméterezhető
- Sorrendek
  - `MethodOrderer.Alphanumeric` - ábécé sorrendben
  - `MethodOrderer.OrderAnnotation` - `@Order` annotáció szerint, mely egy egész számmal paraméterezhető
  - `MethodOrderer.Random` - véletlenszerű

Disabled

- Osztályra és metódusra is tehető a `@Disabled` annotáció
- Nem futtatja le a környezet
- Opcionálisan megadható üzenet
- Ne használjunk commentelést
- Ideiglenesen használjuk

Conditional Test Execution

- Feltételes teszt végrehajtás különböző feltételektől függően
- Pl. operációs rendszer, JVM verziószáma, operációs rendszer környezeti változó, Java környezeti változó
- Pl. `@DisabledOnOs(WINDOWS)`

---

class: inverse, center, middle

Elnevezések

`@DisplayName` annotáció

- Teszt osztályokat és teszt eseteket lehet elnevezni
- Osztályra és metódusra is tehető `@DisplayName` annotáció, melynek paraméterül adandó a megjelenítendő név
- Pl. IDE-ben jelenik meg

```
@Test
@DisplayName("Test the calculation of the age with positive number")
void testGetAge() {
```

```
    // ...
}
```

Display name generation

- Teszt osztály és teszt metódus nevéből generálja (pl. kis- és nagybetűk konvertálása, aláhúzásjelek cseréje)
- Megadható a `@DisplayNameGeneration` annotációval
  - Beépített vagy saját implementáció is használható
- Globálisan is konfigurálható

```
@Test
@DisplayNameGeneration(DisplayNameGenerator.ReplaceUnderscores.class)
void get_age_with_positive_number() {
    // ...
}
```

---

class: inverse, center, middle

Assert

További assert metódusok

- `assertNull`, `assertNotNull` `null` vizsgálatra
- `assertEquals` különböző primitív típusokkal
- `assertNotEquals` annak ellenőrzésére, hogy nem egyenlőek
- `assertEquals` objektumok esetén az `equals()` metódust hívja
- `assertSame`, `assertNotSame` hasonlít össze referenciákat
- `assertTrue`, `assertFalse`

`assertEquals` lebegőpontos számoknál

- Számábrázolási pontatlanságok miatt adjunk meg egy `delta` paramétert

```
assertEquals(1.0, 1.0, 0.005);
```

Assert metódusok több elem esetén

- `assertArrayEquals` tömbök kezelésére
- `assertIterableEquals` `Iterable`, pl. kollekciók vizsgálatára
- `assertLinesMatch` `List<String>` összehasonlítására, de nem csak pontos egyezőséget vizsgál, hanem képes pl. reguláris kifejezésként is értelmezni az elvárt értéket

---

Egyszerre több assert

- Kiértékeli az összes assertet függetlenül attól, hogy sikeres vagy sikertelen
- Mindegyiknek megjelenik az eredménye

```
assertAll(
                () -> assertEquals("John Doe", employee.getName()),
                () -> assertEquals(49, employee.getAge(2019))
        );
```

Alternatív üzenet

```
assertEquals(0, employee.getAge(1970), "Age must be zero");
```

Tesztek gyors lefutása érdekében üzenet kiértékelése csak bukó teszt esetén:

```
assertEquals(1970, getAgeFromXmlDocument(document), () ->
documentToString(document));
```

Assume

- Előfeltételek ellenőrzésére használjuk
- Ha már nincs értelme tovább futtatni a tesztesetet
- Ha elbukik, akkor nem fut tovább a teszt, de nem jelenti hibásnak
- Az assert metódusokhoz hasonló metódusokkal rendelkezik

---

class: inverse, center, middle

Kivételkezelés és timeout tesztelése

Kivételkezelés

- Lambda kifejezéssel paraméterezhető

```
IllegalArgumentException iae = assertThrows(IllegalArgumentException.class,
        () -> new Employee("John Doe", 1800));
assertEquals("Invalid year: 1800", iae.getMessage());
```

Timeout

- `Duration` példánnyal és lambda kifejezéssel paraméterezhető
- `assertTimeout` azonos szálon futtatja

```
assertTimeout(ofMinutes(2), this::longOperation);

String result = assertTimeout(ofMinutes(2), this::longOperation);
```

- `assertTimeoutPreemptively` külön szálon futtatja és megszakítja timeout esetén

```
String result = assertTimeoutPreemptively(ofMinutes(2), this::longOperation);
```

---

class: inverse, center, middle

Egymásba ágyazás

Belső osztályok

- Belső osztályok használatával a tesztesetek között hierarchia építhető fel

```
public class EmployeeTest {

    Employee employee;

    @Nested
    class WithYearOfBirth1970 {

        @BeforeEach
        void init() {
            employee = new Employee("John Doe", 1970);
        }

        @Test
        void testAge() {
            // ...
        }

    }
}
```

Belső osztályok - folytatás

```
@Nested
class WithYearOfBirth2000 {

    @BeforeEach
    void init() {
        employee = new Employee("John Doe", 1970);
    }

    @Test
```

```
    void testAge() {
        // ...
    }

}
```

- Alkalmas kódmegosztásra is, közös inicializáció

---

class: inverse, center, middle

Tagek és metaannotációk használata

Tagek

- Egy teszteset ellátható egy vagy több taggel, mely szerint később szűrni lehet

```
@Test
@Tag("unit")
@Tag("feature-329")
void testGetAge() {
}
```

Filterelés Mavennel

```
<plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.22.2</version>
    <configuration>
        <groups>unit</groups>
    </configuration>
</plugin>
```

- Tag expression adható meg vagy (`|`), és (`&`) valamint nem (`!`) operátorokkal, zárójelek használhatóak (pl. `unit & feature-329 & !long-running`)

Metaannotációk

- JUnit annotációk használata új annotáció létrehozására

```
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Test
@Tag("unit")
public @interface UnitTest {
}
```

```
@UnitTest
void testGetAge() {
}
```

---

class: inverse, center, middle

Dinamikus tesztek

Dinamikus teszt

```
@TestFactory
Stream<DynamicTest> arePalindromes() {
    return Stream.of("racecar", "radar", "mom", "dad")
            .map(text ->
                dynamicTest(
                    "Is palindrome? " + text,
                    () -> assertTrue(isPalindrome(text))));
}
```

Dinamikus teszt elvárt értékkel

```
@TestFactory
Stream<DynamicTest> testGetAge() {
    return Stream
        .of(new Integer[][]{{2000, 0}, {2010, 10}, {2015, 15}, {2050, 50},
{1990, -10}})
        .map(item ->
            dynamicTest(
                "In year " + item[0] + " employee must be " + item[1] +
"years old",
                () -> assertEquals((int) item[1], new Employee("John Doe",
2000).getAge(item[0]))));
}
```

---

class: inverse, center, middle

TempDirectory extension

`@TempDir` annotáció

- Minden teszteset előtt létrehoz ez ideiglenes könyvtárat, és a végén letörli

```java
public class EmployeeWriterTest {

    @TempDir
    Path tempDir;

    @Test
    void testWriteEmployee() {
        Path file = tempDir.resolve("");
        new EmployeeWriter().write(file, new Employee("John Doe", 1970));
        assertEquals("John Doe, 1970", Files.readString(file));
    }
}
```

---

class: inverse, center, middle

JUnit legjobb gyakorlatok

JUnit tanácsok

- POJO tesztelése
- Egyszerű tesztek
- Összes elágazás tesztelve legyen
- Kivételes input adatok tesztelése
- Tesztesetek között ne tároljunk állapotot
- Tesztek futtatási sorrendjére ne alapozzunk
- Dokumentációs célzat
- Lefutásának az idejét minimalizáljuk

Gyakori hibák JUnit használatakor

- A fejlesztő a teszt eredményét manuálisan ellenőrzi
- Egy teszt metódus egyszerre több mindent ellenőriz
- Szükségtelen assertion
- Assert metódus nem megfelelő formájának használata
- Nem teljes lefedettség
- Túl bonyolult teszt eset
- Külső függőségek
- Kivételeket nem hibának deklaráljuk
- Nincs unit teszt
- Test infected

---

class: inverse, center, middle

JUnit 4 és 5 használata

JUnit 4 és 5 együttműködése

- JUnit modulok
  - Platform - tesztesetek futtatására
  - Jupiter - JUnit 5 tesztek írására
  - Vinetage - JUnit 4 támogatás
- Maven - függőségek felvétele

---

pom.xml

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit5.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit4.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.vintage</groupId>
    <artifactId>junit-vintage-engine</artifactId>
    <version>${junit5.version}</version>
    <scope>test</scope>
</dependency>
```

---

class: inverse, center, middle

Hamcrest

Hamcrest

- Harmadik generációs assertek írására un. matcherek használatával
  - Informatívabb hibaüzenet
  - Több lehetőség az összehasonlításra
  - Könnyen használható API

  - Egyszerűbb összehasonlításokból bonyolultabbakat lehessen összeállítani
  - Saját összehasonlításokat lehessen írni
- Matchers anagrammája

Egyszerű ellenőrzés

```
assertThat(employee.getAge(2019), equalTo(40));
// instanceOf, nullValue, sameInstance, stb.

assertThat(pi, closeTo(3.14, 0.005));
// greaterThan, greaterThanOrEqualTo, stb.

// Olvashatóságot könnyítve
assertThat(employee.getAge(2019), is(equalTo(40)));

assertThat(employee.getName(), startsWithIgnoringCase("john"));
// equalToIgnoringWhiteSpace, containsString, endsWith, startsWith
```

JavaBean és logikai kapcsolatok

```
assertThat(employee, hasProperty("name", equalTo("John Doe")));

assertThat(employee, allOf(hasProperty("name", equalTo("John Doe")),
                hasProperty("yearOfBirth", equalTo(1970))));
// anyOf és not
```

- A `hasProperty` használata nem refactor-tűrő

Kollekciók

```
// Pontos egyezőséget néz
assertThat(List.of("John", "Jane", "Jack"), contains("John", "Jane",
"Jack"));
// containsInAnyOrder

// Rész egyezőség
assertThat(List.of("John", "Jane", "Jack"), hasItem("Jane"));

assertThat(List.of(employee), hasItem(hasProperty("name",
startsWithIgnoringCase("john"))));
// hasEntry, hasKey, hasValue Map esetén
```

`pom.xml` függőség

```
<dependency>
    <groupId>org.hamcrest</groupId>
```

```
    <artifactId>hamcrest</artifactId>
    <version>2.1</version>
    <scope>test</scope>
</dependency>
```

---

class: inverse, center, middle

Saját Hamcrest matcher implementálása

Saját matcher implementáció

```
public class EmployeeWithNameMatcher extends TypeSafeMatcher<Employee> {

    private Matcher<String> matcher;

    public EmployeeWithNameMatcher(Matcher<String> matcher) {
        this.matcher = matcher;
    }

    @Override
    protected void describeMismatchSafely(Employee item, Description
mismatchDescription) {
        mismatchDescription.appendText(" employee with name
").appendValue(item.getName());
    }

    @Override
    protected boolean matchesSafely(Employee item) {
        return matcher.matches(item.getName());
    }

    @Override
    public void describeTo(Description description) {
        description.appendText(" employee with name
").appendDescriptionOf(matcher);
    }
}
```

Statikus factory metódus

```
public static Matcher employeeWithName(Matcher<String> matcher) {
    return new EmployeeWithNameMatcher(matcher);
}
```

Használat

```
assertThat(employee, employeeWithName(startsWithIgnoringCase("jack")));
```

- Üzenet hiba esetén:

```
java.lang.AssertionError:
Expected:  employee with name a string starting with "jack" ignoring case
     but:  employee with name "John Doe"
```

---

class: inverse, center, middle

AssertJ

AssertJ

- Fluent assertion
- IDE code complete
- Saját assert írható és generálható

Egyszerű assert

```
import static org.assertj.core.api.Assertions.*;

assertThat(employee.getName()).isEqualTo("John Doe");
// isEqualToIgnoringCase

assertThat(employee.getName()).startsWith("John")
                              .endsWith("Doe");
```

- Fluent

Kollekciók

```
assertThat(employeeNames).hasSize(2)
                         .contains("John Doe", "Jack Doe")
                         .doesNotContain("Jane Doe");
// containsOnly

assertThat(employees).extracting("name")
                         .contains("John Doe", "Jack Doe");
assertThat(employee).extracting(Employee::getName)
                         .contains("John Doe");


assertThat(employees).extracting("name", "yearOfBirth")
```

```
                                .contains(tuple("John Doe", 37),
                                          tuple("Jack Doe", 41));

assertThat(employees).filteredOn(e -> e.getName().contains("a"))
                                .extracting(employee -> employee.getName())
                                .containsOnly("Jack Doe");
```

Üzenetek testreszabása

```
assertThat(employee.getAge()).as("check %s's age", employee.getName())
                             .isEqualTo(40);

assertThat(employee.getName()).withFailMessage("should be %s", "John Doe")
                             .isEqualTo("John Doe");
```

Soft assertion

```
SoftAssertions softly = new SoftAssertions();
softly.assertThat(employee.getName()).isEqualTo("John Doe");
softly.assertThat(employee.getAge()).isEqualTo(40);
softly.assertAll();
```

```
@ExtendWith(SoftAssertionsExtension.class)
public class EmployeeTest {

  @Test
  public void testEmployee(SoftAssertions softly) {
    Employee employee = ...

    softly.assertThat(employee.getName()).isEqualTo("John Doe");
    softly.assertThat(employee.getAge()).isEqualTo(40);
    // Nem kell assertAll() hívás!
  }
}
```

- Összegyűjti a hibákat

Assumption

```
assumeThat(employee.getName()).isEqualTo("John Doe);
```

---

pom.xml

```
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
```

```
  <version>${assertj.version}</version>
  <scope>test</scope>
</dependency>
```

---

class: inverse, center, middle

AssertJ kiterjeszthetőség

Condition

```
Condition<Employee> familyNameIsDoe =
    new Condition<>(e-> e.getName().endsWith("Doe"), "family name is Doe");
assertThat(employee).has(familyNameIsDoe);
// is, not, has, doesNotHave, allOf, anyOf
// are, have, areAtLeast, areAtMost, areExactly - kollekció elemeire
```

Saját assert implementálása

```
public class EmployeeAssert extends AbstractAssert<EmployeeAssert, Employee>
{

    public EmployeeAssert(Employee employee) {
        super(employee, EmployeeAssert.class);
    }

    public static EmployeeAssert assertThat(Employee employee) {
        return new EmployeeAssert(employee);
    }

    public EmployeeAssert hasName(String name) {
        if (!Objects.equals(actual.getName(), name)) {
            failWithMessage("Expected employees name to be <%s> but was
<%s>",
                name, actual.getName());
        }

        return this;
    }
}
```

---

class: inverse, center, middle

Mockito

Mockito

- Test double példányok generálására
- Úgy látszanak, mint az eredeti példány
  - Definiálható, hogy milyen metódushívásra mit adjon vissza
  - Ellenőrizhető, hogy melyik metódus és milyen paraméterrel lett hívva

---

Függőség

UML diagram

UML diagram

Service és Repository

```
public class EmployeeService {

    private EmployeeRepository employeeRepository;

    public EmployeeService(EmployeeRepository employeeRepository) {
        this.employeeRepository = employeeRepository;
    }

    public boolean createEmployee(String name, int yearOfBirth) {
        name = name.trim();
        var employee = employeeRepository.findEmployeeByName(name);
        if (employee.isPresent()) {
            return false;
        }
        else {
            employeeRepository.saveEmployee(new Employee(name, yearOfBirth));
            return true;
        }
    }
}
```

Repository

```
public class EmployeeRepository {

    private List<Employee> employees = new ArrayList<>();

    public void saveEmployee(Employee employee) {
```

```
        employees.add(employee);
    }

    public Optional<Employee> findEmployeeByName(String name) {
        return employees.stream().filter(e ->
e.getName().equals(name)).findFirst();
    }
}
```

Service test mock objektummal

```
@Test
void testSaveEmployee() {
    EmployeeRepository repository = mock(EmployeeRepository.class);
    EmployeeService service = new EmployeeService(repository);
    boolean created = service.createEmployee("John Doe", 1970);
    assertTrue(created);
}
```

Dependency injection

```
@ExtendWith(MockitoExtension.class)
public class EmployeeServiceTest {

    @Mock
    EmployeeRepository repository;

    @InjectMocks
    EmployeeService service;

    @Test
    void testSaveEmployee() {
        boolean created = service.createEmployee("John Doe", 1970);
        assertTrue(created);
    }
}
```

Visszatérési érték definiálása

```
when(repo.findEmployeeByName(anyString()))
                .thenReturn(Optional.of(new Employee("John Doe", 1970)));

boolean created = service.createEmployee("John Doe", 1970);
assertFalse(created);
```

Kivétel:

```
when(repo.findEmployeeByName(anyString()))
    .thenThrow(new IllegalStateException("Constraint violation"));
```

Metódushívás ellenőrzése

```
verify(repo).saveEmployee(any());

verify(repo, never()).saveEmployee(any());
```

Paraméter ellenőrzése

```
ArgumentCaptor<Employee> employeeCaptor =
ArgumentCaptor.forClass(Employee.class);
verify(repo).saveEmployee(employeeCaptor.capture());

assertEquals("John Doe", employeeCaptor.getValue().getName());
assertEquals(1970, employeeCaptor.getValue().getYearOfBirth());

verify(repo).saveEmployee(argThat(e -> e.getName().equals("John Doe")));
```

Generikus típussal:

```
final ArgumentCaptor<List<Employee>> employeeListCaptor
                = ArgumentCaptor.forClass((Class) List.class);
```

---

pom.xml

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>2.25.1</version>
    <scope>test</scope>
</dependency>
```

---

class: inverse, center, middle

Teszt lefedettség

Teszt lefedettség

- A tesztek futtatása során a forráskód mely része fut
  - Utasítások
  - Lehetséges ágak

  - Feltételek
- 70-80%-os lefedettség jónak mondható
- White box testing
- Manuális tesztelésnél is alkalmazható
- IDE
- CI

Jacoco

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>${jacoco.version}</version>
    <executions>
        <execution>
            <id>default-prepare-agent</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

- Java agent alapon
- Gyűjtés a `jacoco.exec` fájlba
- Riport kérése az `mvn jacoco:report` paranccsal

XmlUnit

- XML összehasonlítás
- XPath kifejezések
- Séma validáció

---

class: inverse, center, middle

Integrációs tesztelés Spring Boot használatával

`pom.xml` függőségek

```
<dependencies>

    ...

    <dependency>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.1.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

`pom.xml` pluginek

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.1</version>
        </plugin>
    </plugins>
</build>
```

---

class: inverse, center, middle

Integrációs tesztelés Arquillian használatával

Arquillian

- Integrációs és funkcionális tesztek futtatása eredeti futtatókörnyezet (konténeren) belül
  - Java EE alkalmazásszerver
  - Servlet container (Tomcat, Jetty, stb.)
  - CDI implementáció
  - OSGi konténer

- Tulajdonságai
  - Konténer életciklusának vezérlése (indítás, leállítás, telepítés)
  - Tesztek felruházása új lehetőségekkel (pl. dependency injection)

Célkitűzések

- Legyen hasonlóan egyszerű, mint a unit tesztek futtatása
- Támogassa a konténer cserélhetőséget
- IDE-ből és build eszközből is futtatható legyen
- Létező teszt eszközökhöz integrálható legyen

Működési mód

- In-container működési mód
- Elkészít egy telepítőcsomagot, melyben benne van a teszt eset is
- Telepíti a szerverre
- Konténeren belül futtatja
  - `System.out.println` alkalmazásszerver napló állományában jelenik meg
- Eredményt kijuttatja a teszt futtató keretrendszernek

Teszt eset

- Un. *micro deployment* összeállítása `ShrinkWrap` segítségével
  - Mi kerüljön be a jar, war, ear állományba, mely telepítésre kerül
  - Akár pár komponens (esetleg mock függőségekkel)
  - Kisebb scope - hiba könnyebben azonosítható
- Konténer szolgáltatások elérése
  - Dependency injection (field és teszt metódus paraméterre is)
  - `@Inject`, `@Resource`, `@EJB`, `@PersistenceContext` és `@PersistenceUnit` annotációk

Teszt eset implementáció

```
@RunWith(Arquillian.class)
public class NameTrimmerIntegrationTest {

    @Inject
    private NameTrimmer nameTrimmer;

    @Deployment
    public static WebArchive createDeployment() {
        return ShrinkWrap.create(WebArchive.class)
                .addClass(NameTrimmer.class)
```

```
                .addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml");
    }

    @Test
    public void testTrim() {
        assertEquals("John Doe", nameTrimmer.trimName("  John Doe  "));
    }

}
```

---

pom.xml

```
 <dependency>
    <groupId>org.jboss.arquillian.core</groupId>
    <artifactId>arquillian-core-api</artifactId>
    <version>1.4.1.Final</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.jboss.arquillian.junit</groupId>
    <artifactId>arquillian-junit-container</artifactId>
    <version>1.4.1.Final</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.wildfly.arquillian</groupId>
    <artifactId>wildfly-arquillian-container-remote</artifactId>
    <version>2.1.1.Final</version>
    <scope>test</scope>
</dependency>
```

---

class: inverse, center, middle

Adatbázis réteg integrációs tesztelése Arquilliannal

Adatbázis tesztelés

- Nem unit teszteljük
- Konténerben érdemes a konténer által biztosított szolgáltatások miatt
  - Pl. deklaratív tranzakciókezelés
  - Dependency injection, pl. `@PersistenceUnit`

Idempotencia és izoláltság

- Tesztesetek egymásra hatással vannak

    - Állapot: pl. adatbázis
- Ugyanazon tesztkörnyezeten több tesztelő vagy harness dolgozik
- Megoldás:
    - Teszteset "rendet tesz" maga előtt, un. set-up
    - "Rendet tesz" maga után, un. tear down
        - Rollback?
    - Test fixture
        - Legszélsőségesebb megoldás: adatbázistörlés

---

class: inverse, center, middle

JSF integrációs tesztelése Arquilliannal

DbUnit

Webszolgáltatás tesztelés

SoapUI

Spring integrációs tesztelés

Spring `TestExecutionListener`

Java EE integrációs tesztelés - Arquillian

Concordion

JMeter

- Teljesítmény/terhelésteszt, stressztesz
- Különböző protokollok
- Javaban implementált
- Kiterjeszthető
- Swing felület
- Elosztott tesztelés

- Felvétel

Selenium

SonarQube

- SonarLint

CI

- Jenkins