**Bevezetés a JUnit használatába**

**Unit tesztelés**

**(def, OO, cél, tipikus)**

Az adott programozási nyelv **legkisebb egységét** tudjuk vele tesztelni; ez objektumorientált programozás esetén az osztály

Cél: Az adott egység helyesen oldja-e meg a rábízott feladatot - > az **előírt bemenetre az elvárt kimenetet** adja-e?

tipikusan **automatizáltak** = emberi beavatkozás nélkül futtathatóak

tipikusan a **szoftverfejlesztők írják**

**Mi nem Unit teszt**

**(több osztály, keretrendszer, adatbázis - > integrációs)**

ha **több osztály együttműködését** teszteljük, az nem unit teszt, hanem **integrációs teszt**

ha a teszt futtatásakor **bevonunk valamilyen keretrendszert** (pl. Spring vagy Java EE konténert), az se unit tesztelés

Ha a teszt futtatása közben **adatbázisműveletek** is végződnek, az sem

**jUnit**

**(def, integritás, egység, osztály, eset, metódus, annotáció, Assert-statikus)**

Java programozási nyelven az egyik legelterjedtebb **keretrendszer unit tesztek írására**

Kent Beck és Erich Gamma tette le az alapjait

mindenféle build eszközhöz **integrált** (Ant, Maven, Gradleből is lehet futtatni), és fejlesztő környezetből, pl IDE-ből is

legújabb jUnit 5 – 2017?

Alapvető egysége a java osztály(=**teszt osztály**); ezen java osztályokban **metódusonként** jelennek meg a **teszt esetek**

metódusok ellátása **@Test annotációval,** ezzel jelezve, hogy ezek teszt funkcióval bírnak és futtathatóak közvetlen

**Az elvárt és az aktuális eredmény** összehasonlítása az elnevezés: **Assert**; erre a jUnit **statikus metódusokat** tartalmaz, amivel a két eredmény összehasonlítható

**Tesztelendő kód**



**Manuális ellenőrzés**

Teszt osztály és teszt metódus létrehozása; teszt metódus ellátása a @Test annotációval

szemmel, souttal:



**Given, When, Then**

**(stílus/struktúra, fázisok)**

Teszt eset írásakor érdemes valamiféle **struktúrát** tartani -> egyik ilyen stílus vagy struktúra

a BDD (**Behavior-Driven Developement**) részeként definiálták

fázisok:

Given -> Előfeltételek (teszt eset **előkészítése**, **példányosítás**, beállítjuk a **megfelelő állapotát**)

When -> **meghívjuk** a megfelelő tesztelendő metódust

Then -> leellenőrizzük, hogy a **visszakapott eredmény egyenlő-e az elvárttal**

**asserthez importálás!**

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

**Metódus tesztelés**

****

**Metódus tesztelés egybevonva**

**(egy utasítában)**

Az előkészítés, a példányosítás és az assert **egyetlen utasításként** szerepel, de funkcionalitásban ugyanaz a 3 részt tartalmazza:

****

**Maven integráció**

**(pom.xml dependency test scopeal, surefire plugin, konvenciók a fileok elhelyezésére)**

a jUnit a legtöbb build eszközhöz, így a Mavenhez is integrált

a jUnitot fel kell vennünk **függőségként a pom.xmlbe test scope-al** -> az adott jUnit library nem lesz az alkalmazás integrált része, csak a teszt esetek futtatásához szükséges

**Surefire plugin** használatánál figyelni, hogy csak 2.22tes verziótól támogatja a jUnit 5-öt

konvenció a teszt esetek és erőforrások elhelyezésére a Mavenben:

**src/test/java és src/test/resources** -> így a Maven automatikusan megtalálja őket, lefordítja és **futás közben elhelyezi a classpathon** sőt a tesztek futtatását is elvégzi

**POM.xml módosítás**

jUnit felvétele test scope-al és maven-surfire.plugin beállítás:

****

**IDE integráció**

**(futtatások scopeja, csak elbukott, debug mód, eredmények színekkel és jelentés)**

tartalmaz integrációt a jUnithoz; le lehetet futtatni egy teszt metódust **külön**, vagy akár a **teljes teszt osztály** az összes teszt metódussal egyben, vagy egy **csomag** tesztjeit, vagy **a projekthez tartozó összes** teszt esetet

lehetőség csak az előzőekben elbukott esetek futtatására

Debug mód a tesztekben lévő utasítások lépésekben történő végrehajtására

***Eredmény:***

Passed(zöld): lefutott és az assert is sikeres (elvárt = aktuális)

Failure (sárga) : az assert elbukott

Errror(piros): kivétel, nem várt hiba

**Az első teszteset létrehozása (gyakorlat)**

**Kérdések**

*Mit értünk unit tesztelés alatt és OOP esetén mit tekintünk unitnak?*

*Mi a unit teszt elsődleges célja?*

*Tipikusan milyenek a tesztek és kik írják?*

*Mikor már nem unit teszt a teszt? (3)*

*Mi a jUnit?*

*Mi az alapvető egysége és hogy jelennek meg benne a teszt esetek?*

*Mitől lesz teszt osztály a teszt osztály és teszt metódus a teszt metódus?*

*Mit használunk az elvárt és aktuális eredmény összehasonlítására?*

*Miiért használjuk a Give-When-Then struktúrát és a különböző részek mit definiálnak?*

*Hogyan tudjuk összevonni a 3 fázist?*

*JUnit használatához mit kelll felvenni a pom.xml-be függőségként?*

*Hol kell elhelyezni a teszteket, hogy a Maven automaitkusan egtlaálja őket? Maven integrációhoz milyen plugint kell felvenni a pom.xml-be függőségként?*

*IDEA-s intergációja milyen futtatási módokat tesz lehetővé? Van e debug mód és a tesztek eredményét az Ide milyen szinekkel jelzi?*