**JUnit legjobb gyakorlatok**

jUnit tesztek írásakor érdemes pár szabályt figyelembe venni az implementálásnál:

**F.I.R.S.T elvek**

**(Fast, Isolated, Repetable, Self-Validating, Thorough)**

Fast -> **legyenek gyors lefutásúak** a tesztek; ha lassú, a fejlesztő nem futtatja és értelmét veszti, hiba nem derül ki stb.; fontos, hogy sokszor le legyenek futtatna, és a verziókövető rendszerbe be se kerüljön hibás kód

Isolated/Independent -> **tesztesetek között ne tároljunk állapotot**; az esetek gyorsan és egymástól függetlenül lefuttathatóak legyenek, ne legyenek egymásra hatással, bármilyen sorrendű futás hozza ugyanazt az eredményt; jUnit is támogatja ezt az esetek előtti újrapéldányosíytásával a teszt osztálynak, amivel az atrik és ezáltal az állapot újra inicializálásra kerül; vigyázni kell a statikus attribútumokkal, mert azok megtartják az állapotukat

Repetable -> az esetke akkor jók, ha **bármennyiszeri lefuttatásra is mindig ugyanazt az eredményt** adják vissza

Self-Validating -> **automatizáltak legyenek, ne manuálisan kelljen ellenőrizni** szemmel a fejlesztőnek; tesztesetből **ne írjunk ki konzolra ne naplózzunk, hanem asserteléssel** ellenőriztessük le a kapott eredményeket

Thorough -> **alaposan kell tesztelni**, a tesztelt funkció **összes leágazásával, szélsőértékekkel, nullal, minden eset legyen lefedve;** érdemes arra tesztelni, amikor egyszer sem fut le a ciklusmag, ill. amikor csak egyszer, vagy amikor többször is lefut. A fejlesztő maga írja a tesztet így látja a kódot is és fel tudja építeni a teszteket úgy, hogy átfogóak legyenek

**jUnit további tanácsok**

**(legkisebb scope, egyszerűség, dokumentációs cél, clean code, assertEquals(), egy logikai állítás, de akár összetett vagy több assertbe)**

Az adott eset lehetőleg a **legkisebb scopeot** tartalmazza; ne sok osztályt és azok együttműködését, **hanem csak egy osztályban lévő egy metódus működését** vizsgálja

legyenek **egyszerűek**

használhatóak legyenek **dokumentációs céllal** -> mire használható az osztály? teszteset mutassa meg példákkal

legyen itt is ugyanaz a **clean code elv**!

**assertTrue() helyett assertEquals(),** mert kiírja az elvárt és kapott értékeket

**egyszerre mindig csak egy dolgot, logikai állítást ellenőrizzünk**, ami lehet több assert, de tartozzon egy fogalomkörbe

**Gyakori hibák jUnit használatakor**

**(mindent, assertTrue(), nem teljes->lefedettség mérés, külső függőségek tesztlelése, szélsőségek)**

**mindent le akarunk ellenőrizni;** elég csak a szúrópróbaszerű (pl. összes objektum összes attribútumának ellenőrzésére nincs szükség)

**assertTrue() használata**

**nem teljes a lefedettség**, van ág, amire nincs teszt eset -> erre jók a tesztlefedettség mérő eszközök (zöld sorok ahol le van fedve); lefutott és összes aránya; 80% a célszám

**Külső függőségek használata**, amivel már nem unit teszt a teszt

**Szélsőségek: nincs unit teszt vagy Test infected –** minden letesztelése, 3rd party library, getter és setter; nem cél a 100%os lefedettség, sok energia és idő, miközben fölösleges!