**JUNIT**

Libreria usata su java per fare testing, si può fare in automatico o no, dando un input

Due categorie di test: **black box** vs **white box**

**Differenza**:

* **black box**: non ci interessa come implementato progetto, ci interessa solo se dato un input ci dia l’output desiderato (Ovviamente testare casi limite e/o fondamentali per far funzionare il progetto)
* **white box**: Andremo a testare ogni singolo metodo e classe del progetto

**Si fanno entrambi i test**, in genere prima viene fatto il **black box** nelle grandi aziende, successivamente qualcun’altro testerà facendo **white box**

**UNIT TESTING: u**n white box testing che testa un ‘unità del sorgente

**UNIT:** la porzione più piccola del codice che dovrebbe essere testata

**Divide-et-impera:**

* suddividi il sistema in unità
* ogni unità viene debuggata separatamente
* riduce la probabilità di incontrare bug
* gli errori non si propagano di unità in unità

**JUNIT ANNOTATIONS**

**TEST CASE**: un metodo che verifica speicfiche funzionalità di un unità

**TEST SUIT**: collezione di Unit test

I test case si annotano con **@Test**

* Test method **sono void** e non hanno parametri
* Sono gradite info aggiuntive
  + **@Test**(**timeout**=10): test ok se termina entro 10 secondi
  + **@Test(**expected= illegalArgumentException.class): ok se questa eccezzione viene lanciata
* **@Ignore(“reason”)**: ignora un test
* **@Before**: metodo marcato con before è eseguito **prima di ogni** test case
* **@After**: **dopo ogni** test case
* **@BeforeClass**: metodo invocato all’inizio del test
* **@AfterClass**: metodo invocato alla fine del test
* **@before e @after**:

PARAMETRIZED TEST:

* Usando **@RunWith(Parameterized.class)** e un parametro marcato con **@Parameters**, possiamo eseguire un test su più valori del parametro (Passando una lista o un qualcosa che ha più valori)
* Possiamo fare lo stesso per provare molteplici test, **@SuiteTest**

JUNIT ASSERTS

* assertEquals(expected,actual)
  + funziona con object,int