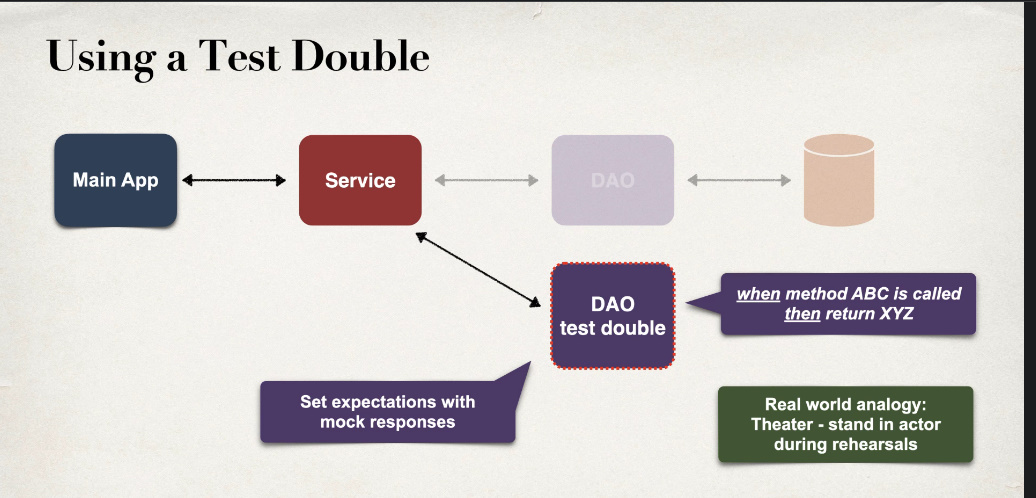
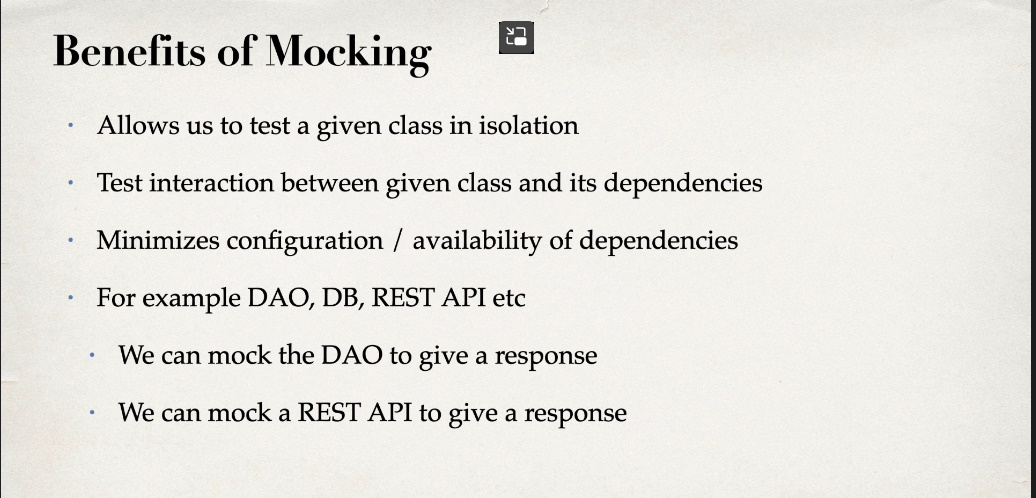
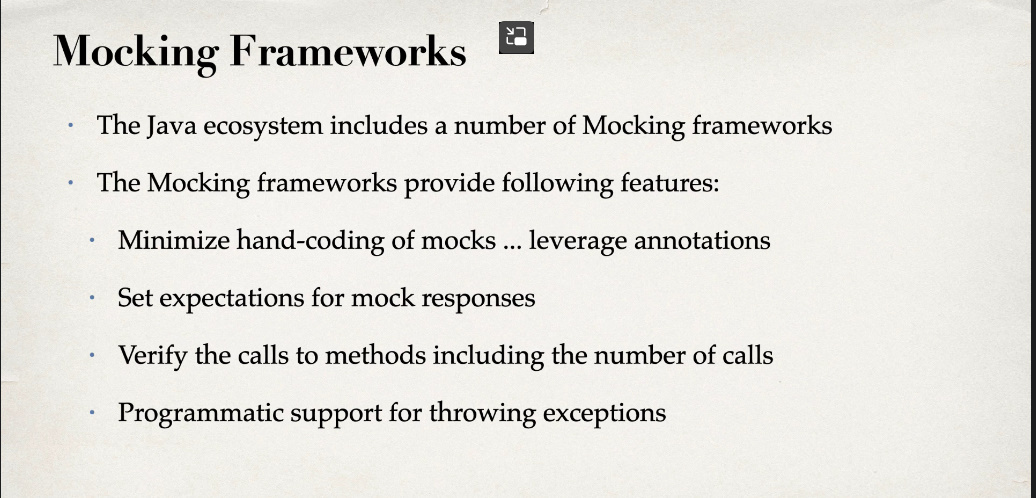


* Junit nu ne poate ajuta cand vine vorba de a testa services
* Testing DB nu mai e un unit test, ci un integration test

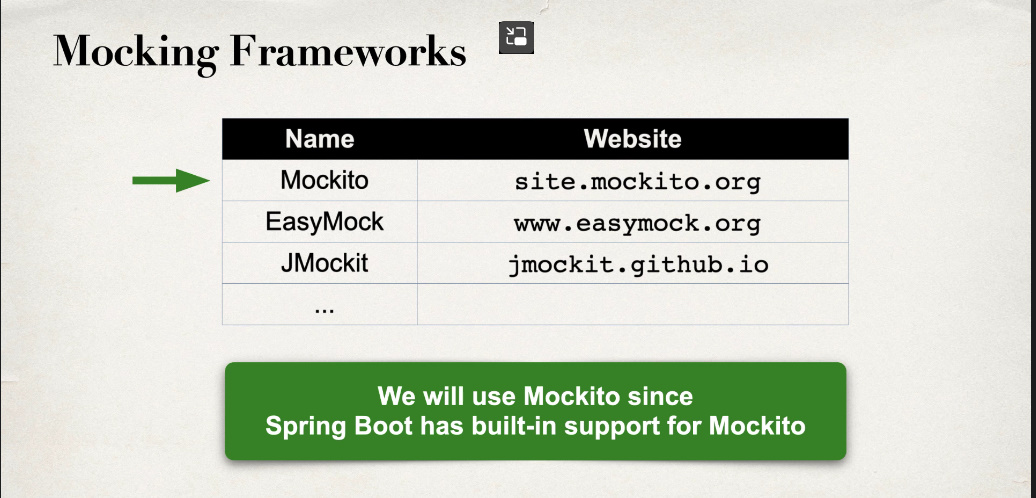
**Test Double(dublura)**

* O solutie este de a folosi un test double
* Aici, avem nevoie ca si cum sa testam ceva in locul la o persoana, care ar interactiona cu baza de date, dar doar vom imita acest proces, nu vom face ceva propriu zis in baza de date
* Test Double se bazeaza pe crearea une dubluri, adica imitarea comportamenutului unei persoane ce foloseste, de ex DAO
* 
* **Tehnica de a folosi test doubles e numita mocking(imitare)**

**Mocking**

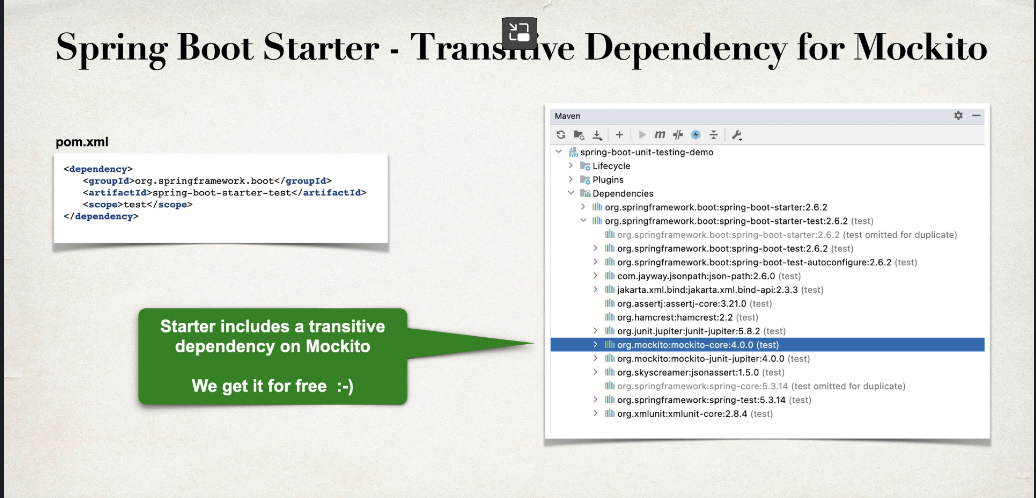
* **Tehnica de a folosi test doubles e numita mocking(imitare),sau mai e numita tehnica de a imita comportamentul unui obiect**
* 
* DAO de ex, nu va comunica propriu zis cu baza de date, doar vom imita acest proces, dand niste rezultate asteptate, deja cunoscute din timp si deci tot setate de noi
* REST API nici nu trebuie sa fie functional propriu zis
* 

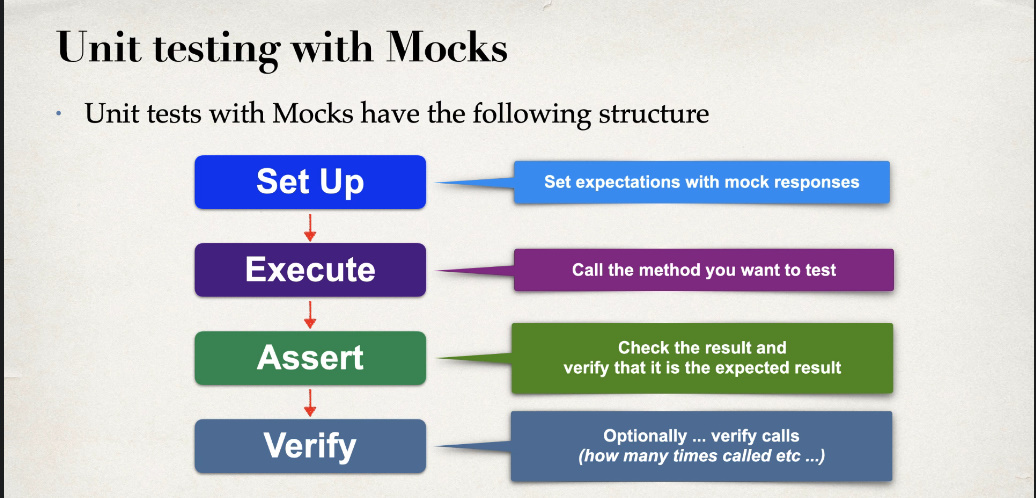
**Frameworks**



**Mockito**

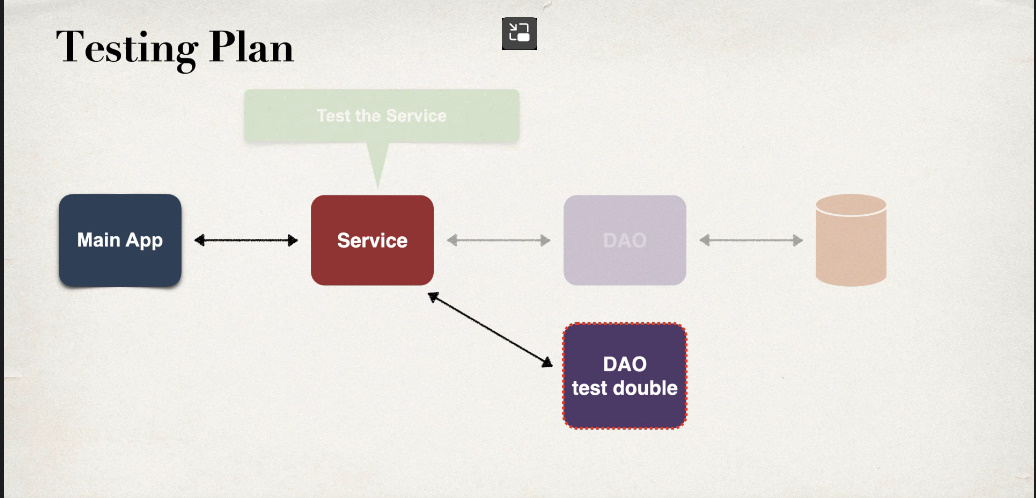
* Spring-boot-test include pe langa Junit si Mockito



* 

Facem Unit tests si cu Mockito.

Tot e necesar sa setam rezultatul asteptat din timp.



**@Mock**

**mock != bean**

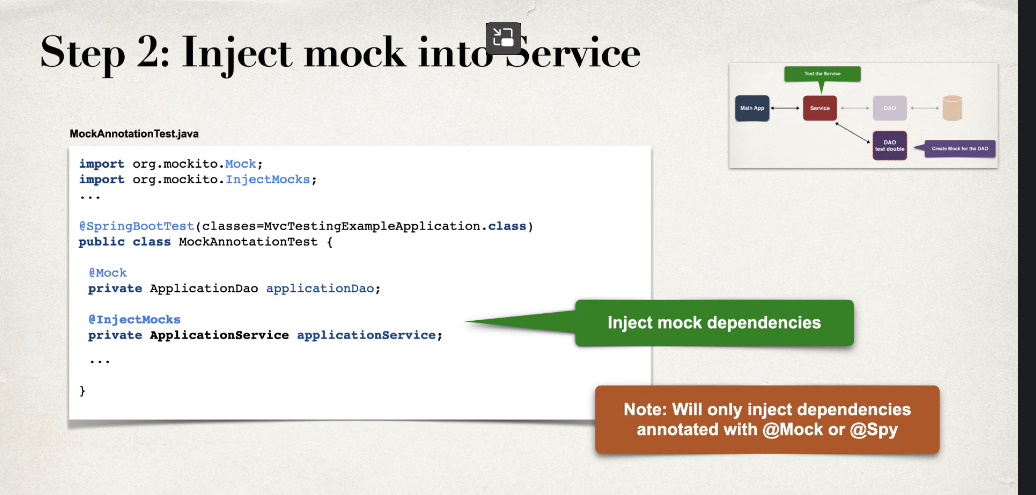
* @Mock este ceva asemanator cu @Bean, doar ca creaza o dublura pentru un obiect, comportamentul caruia va fi mimat mai apoi, numit mock



@Mock nu creaza un bean, ci un mock, ce nu e un bean.

**@InjectMocks**

* **@InjectMocks** face ca toate dependentele necesare, create cu @Mock, sa fie injectate intr-un anumit obiect , dar acest obiect tot e un mock
* **Nu putem oferi o Interfata! Dupa @InjectMocks trebuie mereu sa fie o clasa, dar nu si dupa @Mock. Acolo si o interfata merge.**

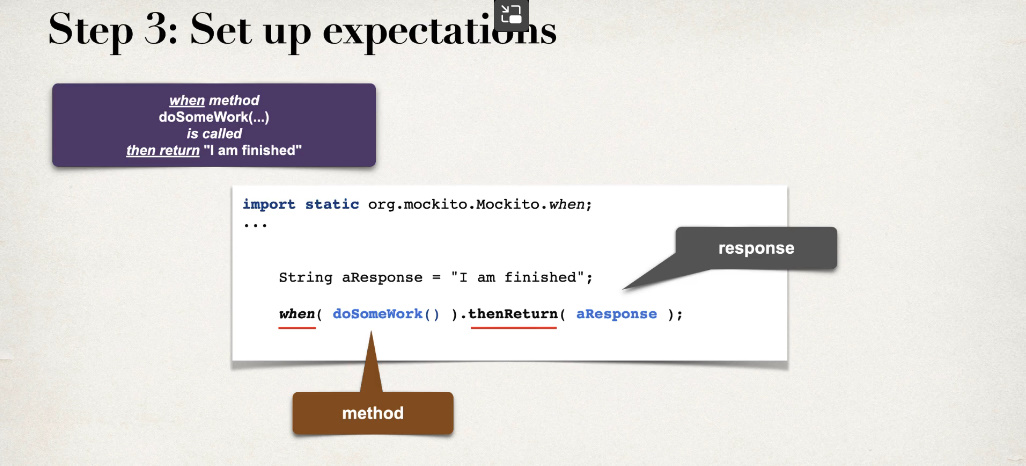
****

**Pot fi injectate doar obiectele ce au @Mock sau @Spy, nu si cele cu @Autowired sau din IoC Container, si asta cam poate fi problema.**

**SetupExpectation**

* Adaugam ce date sa fie returnate cand executam o metoda:

**when(metoda()).thenReturn(“date”)**

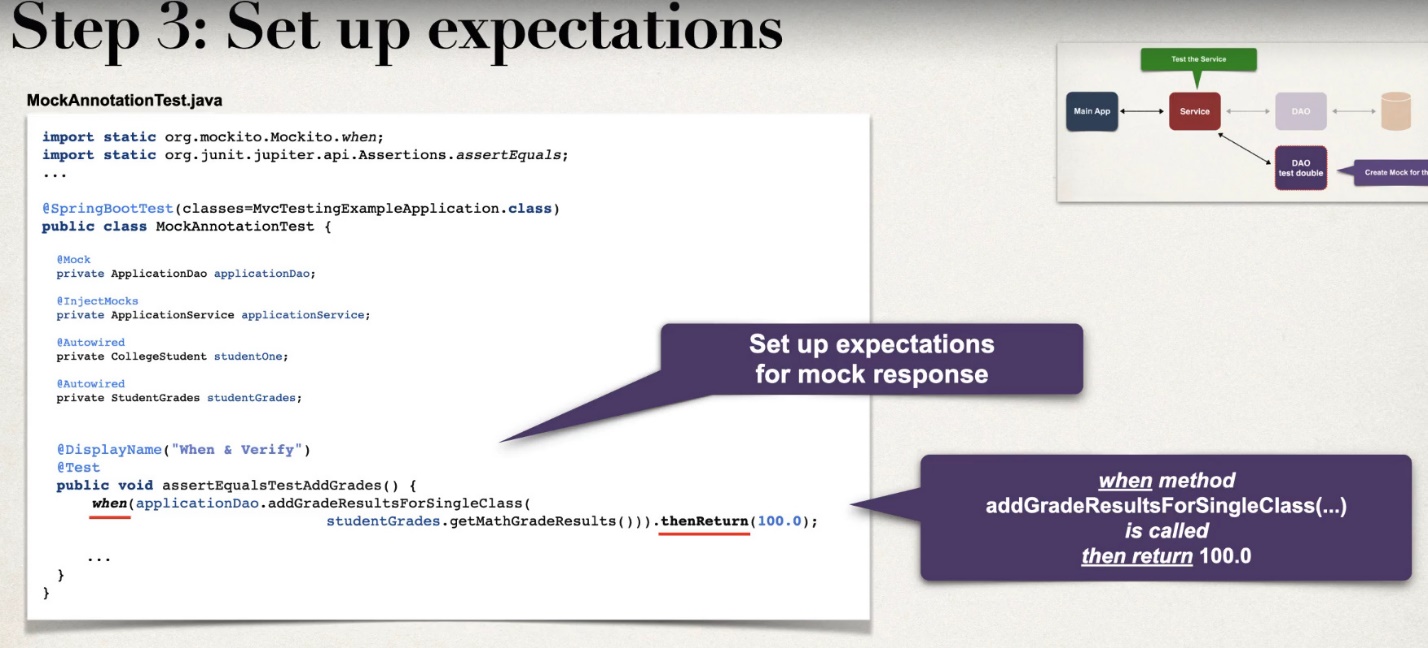


metoda thenReturn() poate returna cam orice obiect, inclusiv si arrays, collection, boolean, int etc. **Atentie! In Spring Boot 3 thenReturn() necesita un parametru de tip Optional! Folosim Optional.of(OurObject)**

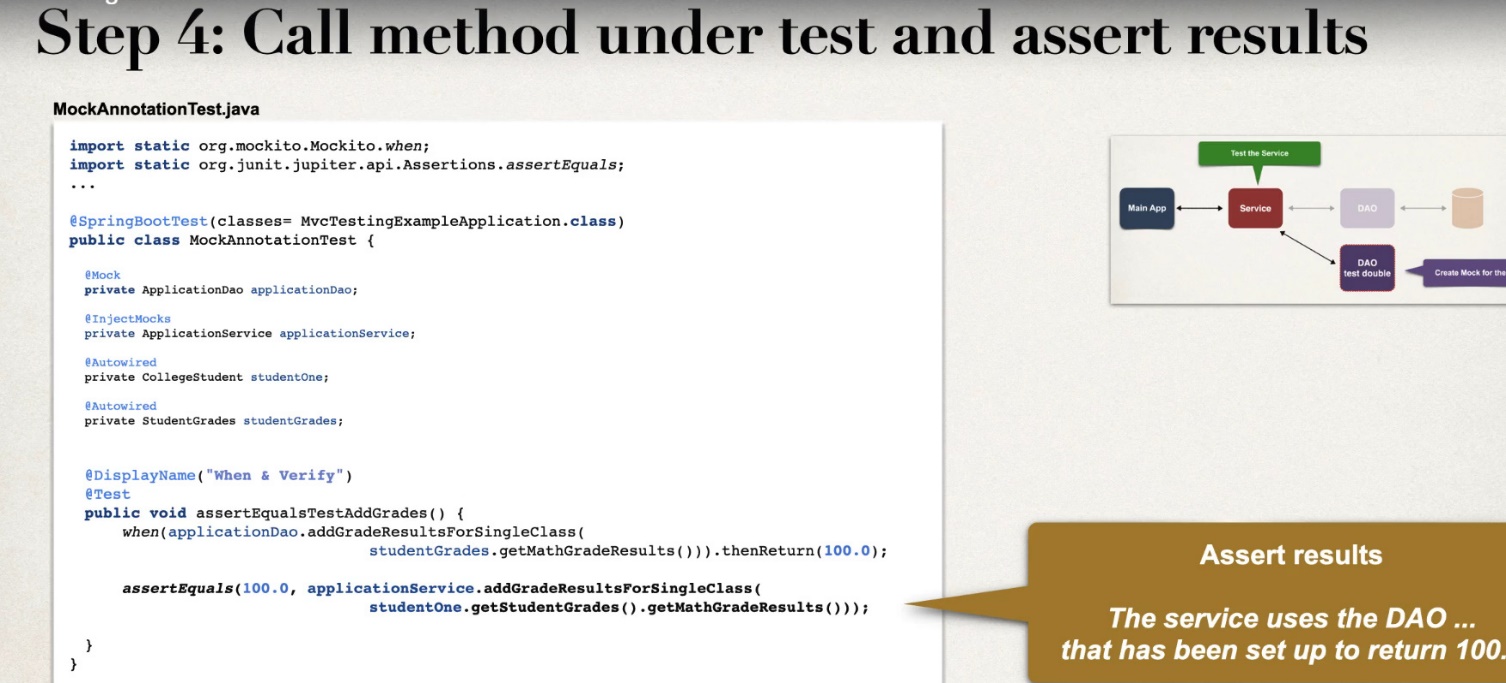
* Asa, setam ce date sa fie returnate la executia metodei.
* Aici nici macar nu se va executa propriu zis metoda din when(), dar cand aceasta va fi rulata altundeva,ca in assert, se va returna acel response setat de noi

**Creare test**

* Cream un test cu Junit exact asa cum facem deobicei, dar vom folosi,pe langa metodele din Assertions, si metode ale lui mockito



* Acum, odata ce metoda din when() va fi rulata, ea nu se va executa propriu zis, ci se va returna direct acea data setata in thenReturn

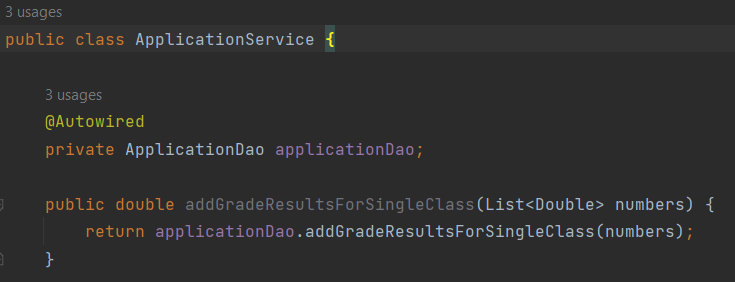




* Apoi, folosim assertEquals sau orice alt assert asa ca deobicei, si cand vom executa metoda din when, se va returna ce am setat noi in thenReturn, asa noi deodata am stabilit ce sa se returneze la executia metodei, si deci ea propriu zis nici nu se va executa.
* **Nu conteaza daca metoda data va fi executata in vreo alta metoda in obiectul cu @InjectMocks.**
* Asa dar, applicationDao se va executa propriu zis, dar cand in metoda addGradeResultsForSingleClass, ar trebuie sa se execute o metoda tot cu acest nume, doar ca a unui bean DAO(asa ar trebuie sa fie,un service are cel putin un dao), ea nu se va executa si se va returna deodata ce am setat noi in thenReturn, adica ca si cum metoda s-ar fi executat si a returnat ea asta, desi noi am mimat totul din timp. Avem grija ca obiectele folosite in when sa fie aceleasi din assert, de altfel nu se va autogenera returnul asa cum am setat.De ex:
* @SpringBootTest  
  class MockitoApplicationTests {  
    
   @Mock  
   Student student1;  
    
   @Mock  
   Student student2;  
    
   @Test  
   void contextLoads() {  
   *when*(student1.getAge()).thenReturn(18);  
   *assertEquals*(18,student2.getAge(),"Wrong age");  
   }  
    
  }



* Avem grija ca sa fie folosite in when() doar obiecte ce sunt mockuri si sunt injectate in altele cu @InjectionMocks, caci de altfel daca am pune un object cu @Autowired, el nici de cum nu va fi injectat cu @InjectionMocks si deci nici nu va fi folosit niciodata.





**verify(mock,times(nr)).metodaMimata()[Optional]**

* Aceasta metoda verifica ca o alta metoda dintr-un mock s-a executat de acel nr de ori, nu mai mult si nu mai putin.
* verify() returneaza referinta la obiectul mock, de aceea dupa verify() executam metoda dorita cu verifiy(...).metodaNoastradDinMock()
* prametrul times(nr) e optional. Fara el, verify() va verifica daca metoda a fost executata odata



