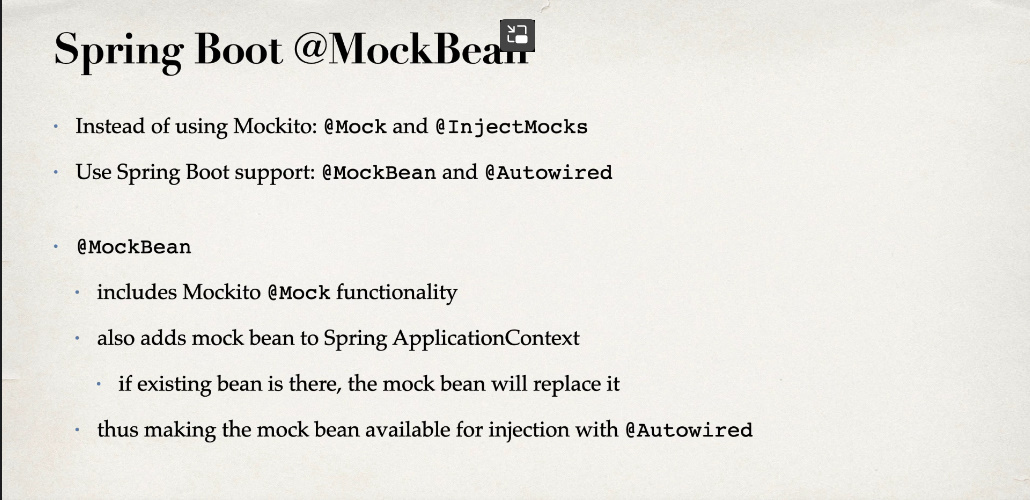


Spring Boot tot ofera suport pentru mockito, deci putem folosi mockito si prin el, in unele cazuri.

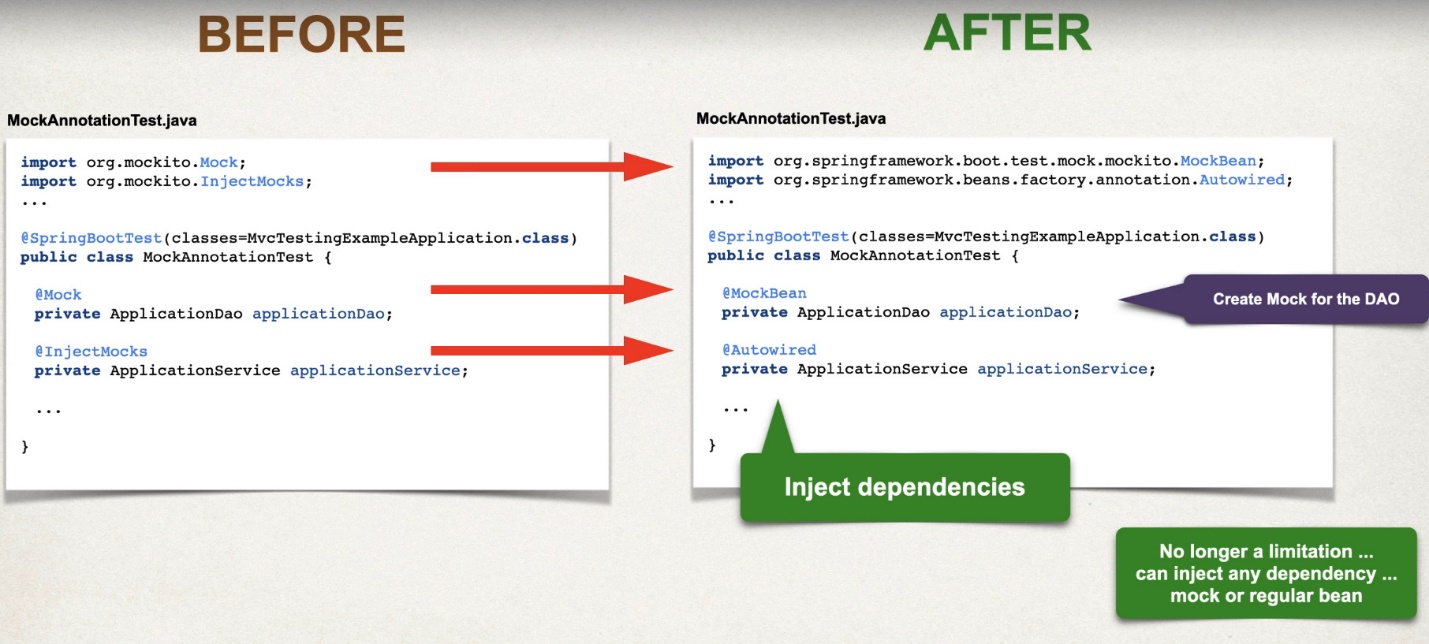
* Cand folosim @Mock si @InjectMocks, apare problema ca @InjectMocks adauga dependente care sunt create de noi cu @Mock si gata. Daca gen obiectul ala are mai multe dependente necesare, si noi am creat doar una cu @Mock, pai doar asta si va fi injectata, si deci beanurile de care mai are nevoie nu vor fi luate din IoC Container, adica unde stau beanurile, ci raman nesetate. @MockBean ne ajuta sa rezolvam problema asta



* Deci, @MockBean e de la Spring, nu Mockito
* Anotatia include tot ce include si @Mock, da mai face ceva:

Creaza un mock ce va fi inregistrat in application context din spring.

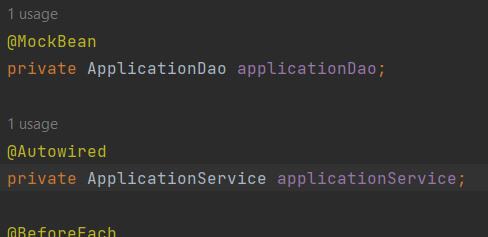
* Daca in app context deja exista un asa bean, el va fi inlocuit cu acest mock bean(bean mimat), daca nu, pur si simplu va fi adaugat acest mock bean.Mock bean nu va fi ca bean original, ci va fi un obiect mimat, fake, ce-i va tine locul.
* Acest bean poate fi folosit in aplicatia spring exact ca si beanurile lui.



In loc de @InjectMocks, putem acum crea un simplu bean cu @Autowired,

si sa il folosim in test.Acum, acest object nu va mai folosi ca dependenta un bean real, dar un bean mimat, creat cu @MockBean, ce l-a inlocuit pe cel original cu unul fake si apoi deja va folosit celelalte beanuri reale de care chiar are nevoie. Nu trebuie sa folosim neaparat mockuri doar. Acest @MockBean va fi inregistrat in IoC container al spring si deci va putea fi injectat in ApplicationService impreuna cu celelalte beanuri de care are nevoie, si beanuri ce poate deja sunt reale, nu mimate doar.

public class MockAnnotationTest {  
 @Autowired  
 ApplicationContext applicationContext;  
  
 @Autowired  
 StudentGrades studentGrades;  
  
 @Autowired  
 CollegeStudent collegeStudent1;  
  
 @MockBean  
 private ApplicationDao applicationDao;  
  
 @Autowired  
 private ApplicationService applicationService;  
   
 @DisplayName(“When & Verify”)  
 @Test  
 public void assertEqualsTest(){  
 *when*(applicationDao.addGradeResultsForSingleClass(studentGrades.getMathGradeResults())).thenReturn(100.0);  
 *assertEquals*(100,applicationService.addGradeResultsForSingleClass(collegeStudent1.getStudentGrades().getMathGradeResults()));  
  
 }





Trebuie sa trimitem neaparat anume obiectele din when():

@DisplayName(“Not Null”)  
@Test  
public void assertNotNullTest(){  
 *when*(applicationDao.checkNull(studentGrades.getMathGradeResults())).thenReturn(true);  
  
 *assertNotNull*(applicationService.checkNull(collegeStudent1.getStudentGrades()));  
}



* E mai bine sa folosim @MockBean si apoi @Autowired, deoarece asa beanul cu @Autowired va primit dependentele sale existente in IoC Controller ce nu sunt mimate, si dependentele pe care le-am mimat, caci ele tot au fost puse in IoC Container, si obiectul ala nu stie de asta. Cu @InjectMocks, se injecteaza doar Mockuri, si nimic din IoC Container, dar cu @Autowired se injecteaza tot ce in Container, si acele beanuri mimate tot sunt deja acolo, si deci si ele vor fi puse.Ca si cum, e ca @Mock si @InjectMocks, doar ca dependentele necreate cu @Mock, tot sunt adaugate din IoC Container
* **Deci Folosim @MockBean cand avem nevoie sa injectam mockuri si beanuri simple, reale.**
* **Mare atentie cand dam @MockBean la un bean ce are si el dependente. Odata ce acesta e doar un bean mimat sau fake, e logic ca dependentele sale nu vor fi injectate, si deci @Autowired din clasa lui nu va functiona pentru el. Aici va trebui @Autowired, nu @MockBean**