JUNIT 4 ve JUNIT 5

JUNIT, Java kodlarının test süreçlerini gerçekleştirmek amaçlı kullanılan bir unit test kütüphanesidir.

Yazdığımız kodun her daim test edilebilme gereksinimi vardır. Test yapmak projenin ileriki süreçlerinden birçok maliyetten özellikle zaman maliyetinden bizi olabildiğince kurtarır.

Unit Test kavramını açıklamak gerekirse eğer; Unit test kodumuzun en küçük iş parçalarının doğru çalışıp çalışmadığını kontrol ettiğimiz durumdur. Çalışan kodlarımız bir input alır ve bunun sonucunda bizlere bir output döner. Yazdığımız test kodları da bize dönen outputların doğru olup olmadığının kontrolünü sağlar. Bu sayede testimiz başarılı olursa yazdığımız method veya kod parçacığı görevini yerine getirmiş demektir.

JUnit 5 ile birlikte JUnit Platform, JUnit Jupiter, JUnit Vintage gibi alt parçalara ayrılarak platform haline gelmiştir.

JUnit 5, Java 8'e uyumlu ve JUnit 4'ten daha sağlam ve esnek olmayı amaçlamaktadır. JUnit 4 ve JUnit 5 arasındaki bazı büyük farklılıklara odaklanacağız.

- Junit 5 sürümü ile birlikte bazı anotosyonlarda değişikliğe gidilmiştir.

Feature	JUnit 4	Junit 5
Declare a test method	`@Test`	`@Test`
Execute before all test methods in the current class	`@BeforeClass`	`@BeforeAll`
Execute after all test methods in the current class	`@AfterClass`	`@AfterAll`
Execute before each test method	`@Before`	`@BeforeEach`
Execute after each test method	`@After`	`@AfterEach`
Disable a test method/class	`@Ignore`	`@Disabled`
Test factory for dynamic tests	NA	`@TestFactory`
Nested tests	NA	`@Nested`
Tagging and filtering	`@Category`	`@Tag`
Register custom extensions	NA	`@ExtendWith`

- Junit 4 ve Junit 5 arasında mimari farkları vardır.

  Junit 4'te bütün her şey tek bir Jar paketi altında bulunur. JUnit 5 ile birlikte mimaride birtakım farklılıklara gidildi ve JUnit 5, JUnit Platform , JUnit Jupiter ve JUnit Vintage olmak üzere 3 ayrı projeden oluşmaktadır.

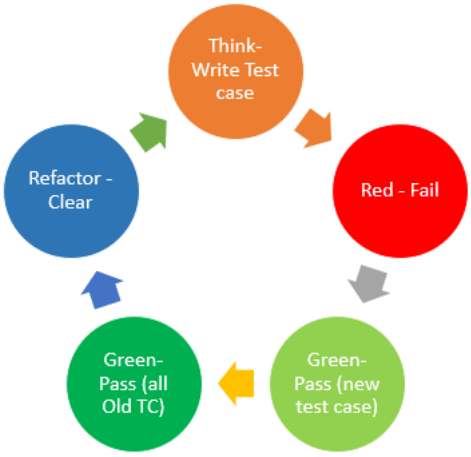
- İhtiyaç olunan SDK sürümleri farklıdır.

  JUnit 4 Java 5 ve üstü bir sürüm gerektirir. JUnit 5 Java 8 ve üstü sürümleri gerektirir.

- Junit 4'te 3. taraf eklentiler ve IDE'ler için entegrasyon desteği yoktur.

  JUnit 5'in bu amaç için özel bir alt projesi vardır, Test oluşturmak için API tanımlar.

TDD(Test Driven Development)

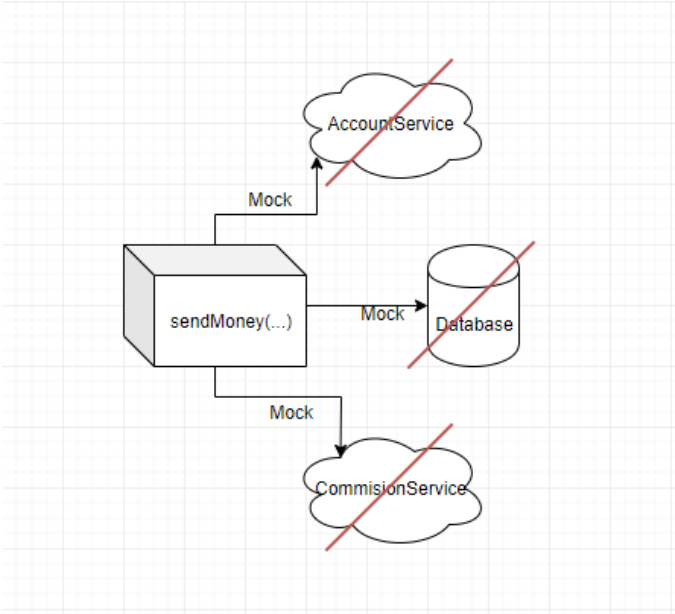


Türkçemize Test Güdümlü Geliştirme olarak geçen TDD bir programlama tekniğidir. Bu programlama tekniğinin esas noktası öncelikle test yazmaktır. Testi oluşturur ve yazarız. Test başarılı olana kadar test etmeye devam ederiz. Testin olumlu olduğu noktada kodumuzu yazarız. Son olarak kodumuzu refactor eder ve süreci tamamlarız.

Test güdümlü geliştirme bazı geliştiriciler için vakit kaybı olarak gelse de projenin gelişimi ile birlikte proje büyüdükçe işlemleri daha hatasız hale getirir. Bakım sürecine giren yazılım için de oldukça faydalı olan bu programlama tekniği bakım maliyetlerini olabildiğince aza indirger.

TDD’nin bizlere sunduğu en önemli kavram KISS (Keep It Simple Stupid) prensibidir. Yani her şeyi basit tutmalıyız. Bu yazılan kodlar ve hatta mimari için de geçerlidir.

Mockito



Mockito tamamıyla ihtiyaç üzerine doğmuş olan bir Unit test kütüphanesidir. Test yazılan süreçte verilere ihtiyacımız vardır fakat her zaman gerçek veriler ile test etme imkanı yakalayamayabiliriz. Bazen de test yazdığımızda yazılan testi gerçek anlamda birim üzerinde uygulamak istememizdir.

Test yazdığımız süreçte veritabanına gidip veri çekmemize gerek yoktur. Çünkü Mockito aracılığıyla Mock yani sahte datalar üretiriz. Yaptığımız test database işlemlerinde soyutlanır ve elimizde bulunan sahte veriler üzerinden işlemlerimizi gerçekleştiriz.

Mocklama yaparken Mockito’yu tercih etmemizin nedeni, Mockito’nun bizim yerimize objelerimizi kendisinin mocklayacak olması aksi takdir de her bağımlı olduğunuz kaynak için bizim somut bir mock objesi oluşturmak zorunda kalacağımız idi.

SPRING PROFILE

Yazılımlarımız için bazen farklı konfigürasyonlar gerekmektedir. Projemizin hem DEV hemde CANLI ortamlarda çalışan iki versiyonu olabilir. Bu süreçte kod içerisinde tanımlanan database url i projenin hangi portta çalışacağı veya tanımlanacak diğer özellikleri de farklılık gösterir.

Spring profile bizler için yapacağımız konfigürasyonlarda yardımcı olur. application.properties / application.yml dosyalarına konfigürasyonlarımızı yazarak daha sonra bunları ActiveProfiles anotasyonu sayesinde seçebiliriz.

Örneğin;

```
@ActiveProfiles("dev")
public class TestClass{
 ...
}
```

Burada ActiveProfiles anotasyonunda properties dosyasında tanımlamış olduğumuz profile ismini tanımlıyoruz ve işlem tamam.