**Test Driven Development**

**What is TDD ?**

Software development approach that you create test cases to validate actual code.

Yazdığımız kodu doğrulamak için test senaryoları oluşturduğumuz yazılım geliştirme yaklaşımı.

**3 Laws of TDD**

* Don’t write production code without a failing test
* Write only enough test code as is sufficient enough to fail test
* Implement minimum needed code to make test fail.

İlk madde örnek üzerinden bakacak olursak bi siparis olusturma, veya bitirme süreci olsun ilk önce testini yazıyoruz, ilk fail alana kadar prod kodu yazmıyoruz ve sonradan eklemeye baslıyoruz.

İkinci madde ise mesela siparis olusturma kodunu yazdık, diyelim ki testi 1 parametresi ile gecirdik prod kodunu da ona göre yazdık, sonra 2 parametresini gecince test fail verdi o zaman gidip prod kodunu da buna göre degistiriyoruz. Yani T**estte başarısız olmak için yeterli olduğu kadar test kodu yazmalıyız.**

Ucuncu madde ise prod tarafına eklediğimiz kodda bir adım test yazıp sonra bir adım prod kodu sonra bi adım test, bir adım prod kodu bu sekilde adım adım ilerlemek her zaman daha iyidir. Zaten ikinci adımda olmasa ucuncu adımda test fail olacağı icin prod kodunu da bastan sona düzeltmek yerine kaldığımız yerden düzeltiriz iterasyon gibi. (Testin başarısız olması için gereken minimum kodu uygulayın)

**Advantages of TDD**

* Modular and Testable Code
* Easy to Maintain
* Smooth refactoring
* Maintenance cost reduction
* Eliminate Debugging
* Tests are the documentation of code

Bir kodun testini yazamıyorsak o kod hatalıdır, burada testini yazdığımız kodda moduler ve test edilebilir olmalı ve mümkün oldugunca o modül icinde kalınması fayda sağlar.

Bakım gibi düsünebiliriz, herhangi bir bug geldiğin de test oldugu icin daha kolay cözülebilir.

Refactor kısmında kodu degistirdiğimiz de orada herhangi bir mantığı bozduysak test kısmında bu bozulmayı anlayabiliriz.

Normalde testi olmayan kodda prod ortamıdna patlaması normal, cünkü parametre degistiğin de test olmadıgı icin patlaması normal. Yazmayı TDD ile yapıyorsak bakım oranı da düsük olacaktır.

Bir problemi cözmek icin debug’tan ziyade testimiz olursa prod ortamında alınan hatanın aynı case’ni test kodunda calıstırıp denediğimiz zaman daha da hızlı ilerlemis oluruz.

**Disadvantages of TDD**

* Slows down development process (in the beginning)
* TDD approach is hard to learn ( in the beginning)
* Hard to maintain especially within strict deadlines
* All the team members should use it

Bildiğimiz bir konu alanında gelistirme yaparken mesela siparis olusturma, onu direkt yazabiliriz aslında ama bir test tarafına bir prod tarafına kod yazdığımız zaman bu başlangıcta bizi yavaşlatıyor.

TDD yaklaşımını öğrenmek zordur. (Baslangıcta)

Prod ortamında problem oldu diyelim, o stres düzeyinde bir test bir prod ortamında degisiklik yapmak kolay olmuyor ve son tarihlerde bakımı zor oluyor. O yüzden hard to maintain.

Takımda bir kisi TDD yapıyor geri kalanı yapmıyorsa sıkıntı, o yüzden Tüm üyeler TDD kullanmalı.

**Unit Testing**

Testing individual modules in isolation without dependencies to confirm code works.

Kodun çalıştığını onaylamak için bağımlılıklar olmadan tek tek modülleri izolasyon olarak test etmeliyiz.

Herhangi bir özellik successful order creation bu bir flowdur. Aynı zamanda da requirement bir listedeki eleman gibi düsünebiliriz. Siparis bilgilerini girdiğimiz de siparisin verilmesini istiyoruz. Bu bir istek. Gerekliliklerden bi tanesi. Burada siparisin verilmesi kısmı ana logic.

Bu siparisi verirken depend olduğumuz baska bilesenler de olabilir. Odaklanmamız gereken yer ana logic ( order creation) dolayısıyla tek bir yere odaklanınca depend olunan diğer bileşenleri varmış gibi kullanım sağlayabiliyoruz. (Mock)

Diyelim ki siparisin delivery(teslim), payment(ödeme) ye bağımlılığı var. Payment success oldugunu düsünüp, delivery de aynı sekilde de düsünüp order creationun basarılı sekilde yapılıp db ye kayıt edilebilir dediğimiz zaman delivery ve payment detaylarına girmemis oluyoruz. Sadece bir modulün icindeki ana logici bağımlı oldugu diğer bilesenleri düsünmeden o modülün kendi icinde izole bir sekilde test edebiliyoruz. **Buna unit Test denir.**

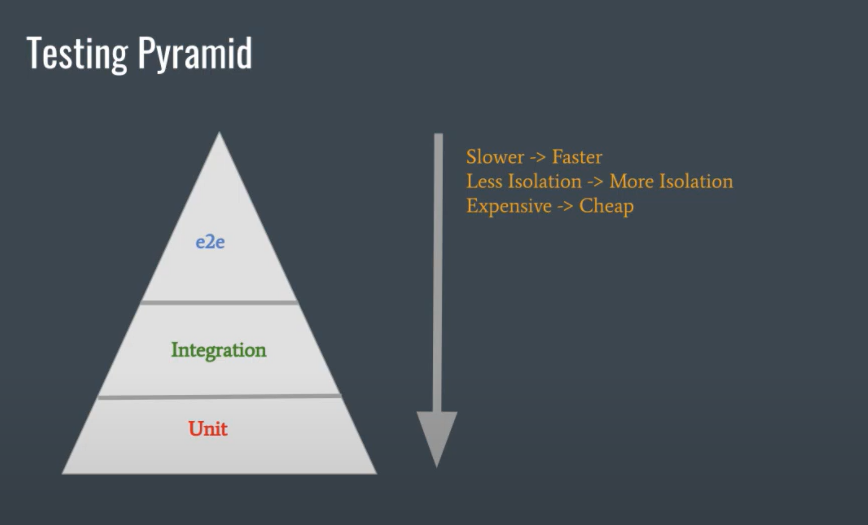
**Integration Testing**

Checking multiple modules together to see they work correctly.

**Doğru çalıştıklarını görmek için birden fazla modülü birlikte kontrol etme.**

Bi tane modül yani A modülü B modülü ile birlikte calısabiliyor mu gibisinden testlere diyoruz.

**Not** : Mesela 2 tane unit test yazdık ama 2 modül bir arada calıstırınca patlıyor burada da **Integration Testing eksikliğini görüyoruz.**



e2e testi ise sistemin genelini test eden yapıdır. Yani UI üzerinden payment API’nin test edilmesi gibi. e2e en yavaş çalışan testidir. En hızlı da Unit.

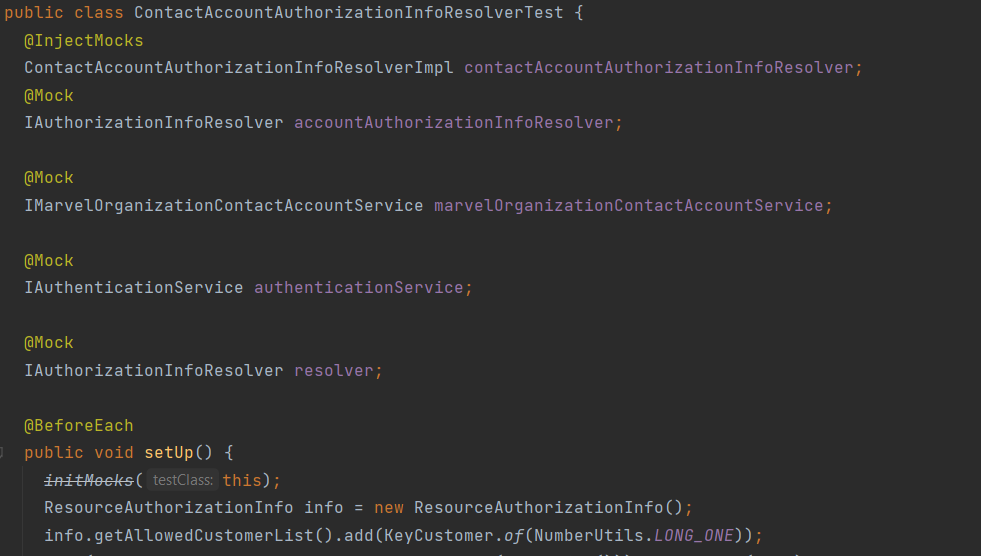
**MOCKITO**

Mocking is to isolate the behaviour of dependency and focus on the part that being tested

Mockito is mocking library in Java

Varmıs gibi davranması, bağımlılığın davranışını izole etmek ve test edilen parçaya odaklanmaktır.

Yukarıda dediğimiz gibi mesela order creation ana logicimiz olsun. Bunun depend oldugu payment ve delivery sürecleri var bu dependlerin ic logicine girmeden bunları varmıs gibi kullanıp direkt ana logice odaklanıp testi yapabiliriz. **Burada varmıs gibi durumunu bize sağlayan library ise Mockito’dur. Asagıda örnek bir sekli verilmistir.**

****

**Argument Capture**

It allows us to capture the argument passed to a method which we don’t have direct access.

Doğrudan erişimimiz olmayan bir yönteme iletilen argümanı yakalamamızı sağlar.

Test ederken bir fonksiyona parametreler gönderiyoruz, o fonksiyon icinde onun baglı oldugu baska yerler olabilir, mesela productService, kendi icinde catalogService bağımlılığı var. Yani burada productService gectiğimiz parametreye karsılık, catalogdaki parametrelerde x parametre ile calıssın gibi bir mantık ile yapabiliriz. Bunu da **Argument Capture**

Sağlar.

**Integration testlerde db olarak embedded db kullanmak faydalıdır. Bir seyi mocklamıyoruz. H2 database tarzında ya da test container kullanarak calısma anında kullanıp uygulamayı kapattığında ölen db’ler ile yapılması mantıklıdır.**

BDD’de user olarak onun testini yapıyoruz, TDD’de birim test yapıyoruz.

Fonksiyonel test , sistem seviyesinde testler denir. **Uctan uca test**.

Unit testte kücük bir birimi test edersin. **Birim test**