# Obje Mocklama(Mockito)

## Obje Mocklama(Mockito) Nedir?

Mocking yazılan kodu test ederken bağımlı oldunulan objelerin fake hallerinin oluşturulup kodun test edilmesine denir. Bunu yapılmasının nedeni birim test yaparken, testi gerçek anlamada birim düzeyinde yapmaktır. Buna birim testlerinde ihtiyaç duyulmasının temelde iki sebebi vardır:

1. Birim testleri, genelde test ettikleri sistemin kendisi ile ilgili varsayımları doğrulamak için yazılır.
2. Test dublörleri, davranış ve kullanım şekillerine göre çeşitlenir. Bunlardan en çok kullanılanları dummy, fake, stub, spy ve mock'tur denebilir. Bu çeşitliliğe sebep olan genel faktörler, bu dublörlerin beklenen işi yapıp yapmadığı ve yaparken nasıl bir davranış gösterdiği ile ilgilidir.

Mock'ların genel kullanım şekli, yerine geçtiği bağımlılık üzerinde, test edilen sistemin yapması beklenen işlemlerin yapılıp yapılmadığını doğrulamak olarak tanımlanabilir. Mock nesnelerinin casus nesnelerinden (spy) farkı, her ikisi de üzerlerinde yapılan işlemleri takip ederken, mocklar bu işlemi testlerin doğrulama (assert) kısmına da entegre ederler.

Mocklama işlemi genelde kütüphaneler yardımıyla, test metodlarının içinde, veya tekrar eden test ayarlarının yapıldığı bir özelleştirme metodunu, satır arası kodlar ile yapılır. Yani çoğu zaman mocklanan tipden devralan bir tip yazılmaz. Mock kütüphaneleri genelde bu işi, dilin reflection kütüphanesinden faydalanarak, çalışma zamanında, ayarlanan kurulumu sağlayacak vekil tipler üreterek sağlarlar. Bunun getirdiği bir avantaj, bütün ön koşulları yerine getiren bir dublör birimi yazmadan, her test metodu için yalnızca beklenen davranışı sağlayan satır içi ayarlamalar yapılmasına olanak sağlamasıdır.

Mock kütüphanelerinin diğer bir avantajı ise, genellikle kendi içlerinde doğrulama (assertion) mekanizmaları bulundurmaları ve mocklanan nesne üzerindeki beklentilerin karşılanıp karşılanmamasına göre, çağırıldıkları testin başarı durumunu etkilemeleridir. Mock kütüphanelerinin yaptığı bu işi, elle yazılan mock tiplerinde geliştiricinin kendisi yapması gerekebilir.

Mocklama işlemi yapılırken genelde bağımlı olan birimle ilgili yapılması beklenen çağırmalarla ilgili ayarların yapılmasının yanı sıra, bazen test edilen sistemin doğru çalışması için, bu çağırmaların geriye değerler döndürmesi gerekmektedir. Bu durumları çözmek için, geliştirilen veya ayarlanan mock tipinin, test edilen tipi rahatsız etmeyecek bir değer döndürmesi gerekebilir. Mock kütüphaneleri böyle durumlarda yapılan ayarlamalarda geriye değer döndürmeye de olanak sağlar.

### Dezavantajlar

- Test ortamında mock kullanılacak bir sistemde neredeyse her şeyin birer arayüz (interface) üzerine inşa edilmesi gerekebiliyor. Bu da kimi zaman over-engineering olarak nitelendirilen probleme yol açabiliyor. Arayüz gerekmeyen yerlerde sınıfların kullanılması da sadece mocklamanın yapılabilmesi için normal şartlarda son ve kesin halinde olan metodların (örneğin C# için), virtual olarak işaretlenmesini gerektirebiliyor. Sınıflardan devralarak yazılan mocklar için bir başka sorun ise constructor'ların her durumda çağrılması, örneğin veritabanına constructor'ında bağlantı sağlayan bir sınıfın üzerine inşa edilen bir mock nesnesi de kullanmayacağı halde yine bu bağlantıyı kurmak zorunda kalabiliyor.

- Test ortamını kimi zaman karmaşıklaştırması olabilir. Bağımlı olunan sınıf yirmi satırlık bir kod barındırabilirken, test ortamının gerekliliklerini sağlamak için yüzlerce satır kod ile bahsedilen sınıfı taklit eden bir mock objesi yazmak gerekebiliyor. Bu karmaşıklığı çözmek için, "kendini tekrar etme" prensibini kullanmak gerekebiliyor. Bunun için test metodlarında parametreler ve bu parametreleri çözümleyen "fixture" kütüphanelerini kullanılabilir. Örneğin, .Net Framework için AutoFixture kütüphanesi, AutoMoq adında bir alt kütüphane ile, test metodlarına yer alan arayüzleri Moq kütüphanesinin Mock nesnelerini kullanarak çözümleyebiliyor.

Java Mock Kütüphaneleri

JMock: https://github.com/jmock-developers/jmock-library

Mockito: https://github.com/mockito/mockito

EasyMock: https://github.com/easymock/easymock

Çoğu mock kütüphanesi, mocklama işleminin çokça yapıldığı durumlarda işe yarar hale geliyor. Eğer test ortamı veya test edilen sistem küçükse, elle yazılan mocklar çoğu zaman işi görüyor. Ancak sistem büyüdükçe, production kodu yazarcasına mock nesneleri yazmak, zahmetli hale gelebiliyor, böyle durumlarda kütüphanelere başvurmakta fayda var. Mock kütüphanelerinde ön plana çıkan farklılıkları şöyle listeleyebiliriz:

- Test Framework'ü ile uyumları
- Fixture edilebilir olmaları
- Virtual olmayan sınıfları ve/veya statik metodları mocklayabilirmeleri

Bunun dışında performans ve yazılış ile ilgili farklılıkları da ortaya çıkabiliyor. Test edilen sistemin ve bağımlı olunan tiplerin özelliklerine göre kütüphane seçmekte -veya seçmemekte- fayda var.

Yararlanılan Kaynaklar;

https://devnot.com/2018/mocking-nedir/#:~:text=Mocking(mocklama)%2C%20pop%C3%BCler%20yaz%C4%B1l%C4%B1m,(test%20double)%20olarak%20tan%C4%B1mlanabilir.