Unit Test

1. .NET'te Unit Test Yapma:

* .NET platformunda unit test yapma sürecini açıklayın ve örnek olarak Xunit ve Moq kütüphanelerini kullanarak bir unit test yazın.

1. Xunit ve Moq Temel Kavramları:

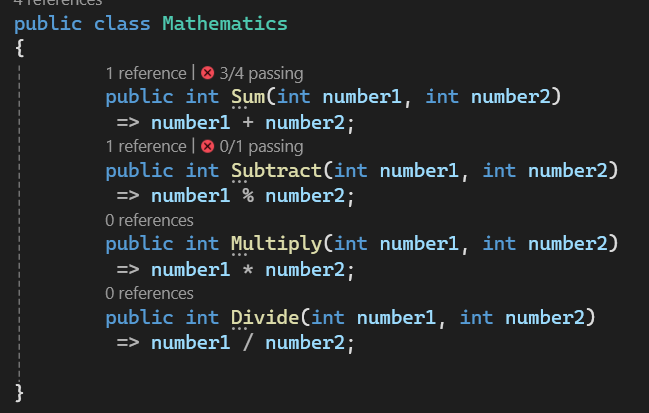
* Bu kütüphanelerde sıkça kullanılan fonksiyonları ve kavramları açıklayın. Özellikle aşağıdaki kavramlara odaklanın:
* Mocked object üretme: Testlerde bağımlılıkları izole etmek için nesneleri nasıl "mock"layabiliriz?
* Assert işlemleri: Beklenen ve gerçek değerleri doğrulama yöntemleri.
* Fact ve Theory: Farklı test senaryoları için Fact ve Theory kullanımını kod üzerinde uygulayalım.

**Unit test**, bir yazılımın en küçük birimlerini bağımsız olarak test etme işlemidir.

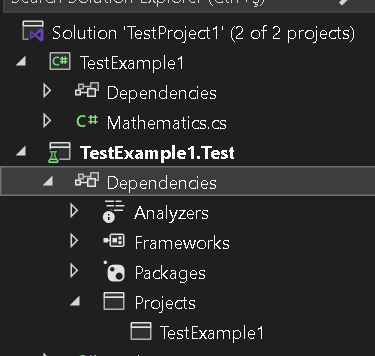
.NET ’de Unit Test yapabilmek için öncelikle test edilecek bir uygulamaya ihtiyaç vardır. Ardından ilgili uygulamayı test edebileceğimiz ve kuralları koyup değerlendirebileceğimiz bir test projesine ihtiyaç vardır.  Test için kullanılacak framework’lere ihtiyaç vardır. Bunlardan biride xUnit.Net Framework’üdür.

**xUnit .Net projesi oluşturma**

Test edilecek bir matematik işlemleri içerin uygulamamız var.



Xunit test projemizi oluşturup test edilecek projeyide ekliyoruz. Proje referansınıda ekliyoruz test kodlara erişebilsin diye.



Unit test yazmak Arrange, Act ve Assert olmak üzere üç aşamadan oluşur.

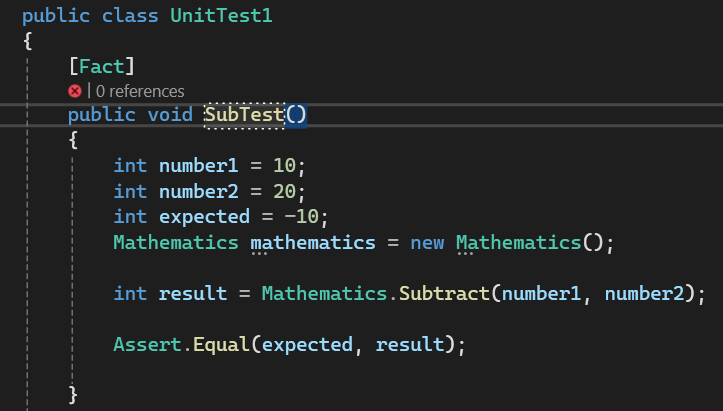
* ***Arrange***  
  Test edilecek metodun kullanacağı kaynakların hazırlandığı bölümdür. Değişken tanımlama, nesne oluşturma vs. gerçekleştirilir.
* **Act**  
  Arrange aşamasında hazırlanan değişkenler yahut nesneler eşliğinde test edilecek olan metodun çalıştırıldığı bölümdür.
* ***Assert***  
  Act aşamasında yapılan testi doğtulama aşaması. herhangi bir türde result dönebilir.

**Test örneği**

 ‘Mathematics’ sınıfındaki metotları test edebilmek için ‘Test’ projesinde bir ‘MathematicsTest’ isminde bir class oluşturup ‘Subtract’ metodunu test edelim.



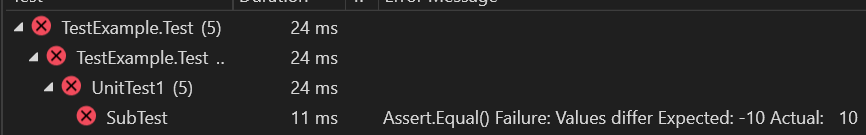
‘Mathematics’ sınıfındaki ‘Subtract’ metodu;



‘unitTest1’ sınıfındaki ‘Subtract’ metodunu test eden ‘SubTest’ metodu

parametre almadığı için ‘Fact’ attribute’u ile işaretli.

Test Explorer’, ‘Fact’ ile işaretlenmiş metotların test metotları olduğunu algılamakta ve içerisindeki ‘Assert’ değerlendirmelerine göre ilgili testin sonucunu aşağıdaki gibi bizlere sunmaktadır.

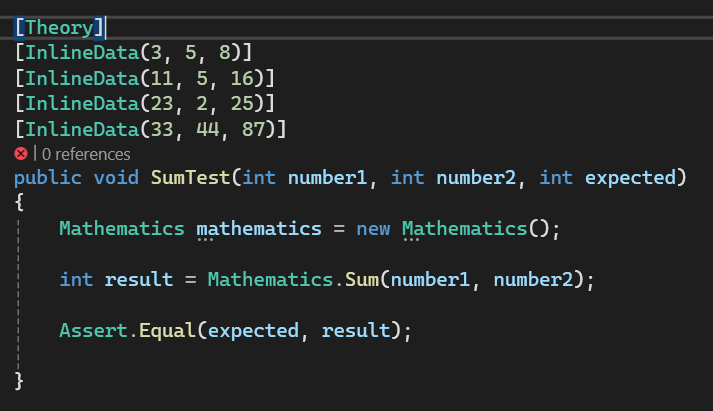


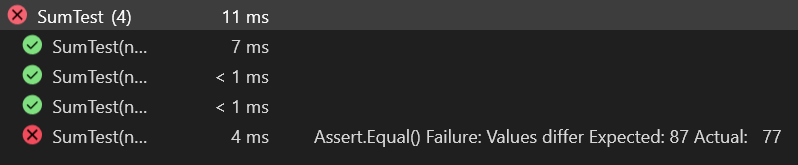
Expected ‘-10′ değeri iken actual ’10’ değeri aldı . Dolayısıyla bu metot(SubtractTest) testten geçemedi.

Assert fonksiyonunun kontrol edebildiği durumlar

|  |  |  |
| --- | --- | --- |
| Fonksiyon | Açıklama | Örnek Kullanım |
| Equal/NotEqual | Sonuçla bekleneni kıyaslar. | Assert.Equal(expected, result); Assert.NotEqual(expected, result); |
| Contain/DoesNotContain | Sonuç içinde değer arar. | var containsValues = new[] { 3, 5, 7, -10 };  var doesNotContainsValues = new[] { 2, 4, 6 };  Assert.Contains<int>(values, v => v == result); Assert.DoesNotContain<int>(values, v => v == result); |
| True/False | Şartın doğru/yanlış olduğunu kontrol eder. | Assert.True(result < 10); Assert.False(result > 10); |
| Match/DoesNotMatch | Regex ifadesine uyum kontrolü yapar. | Assert.Matches("sa{2}t", "saat"); Assert.DoesNotMatch("sa{2}t", "muiddin"); |
| StartsWith/EndsWith | Başlangıç/bitişi kontrol eder. | Assert.StartsWith("G", "Gençay"); Assert.EndsWith("y", "Gençay"); |
| Empty/NotEmpty | Koleksiyonun boş olup olmadığını kontrol eder. | var emptyCollection = new List<object>();  var notEmptyCollection = new List<object>() { 3 };  Assert.Empty(collection); Assert.NotEmpty(collection); |
| InRange/NotInRange | Değerin aralıkta olup olmadığını kontrol eder. | Assert.InRange(result, -1000, 1000); Assert.NotInRange(result, 1000, 2000); |
| Single | Koleksiyonda tek bir veri olup olmadığını kontrol eder. | var collection = new List<object> { 3 };  Assert.Single(collection); |
| IsType/IsNotType | Değerin türünü kontrol eder. | Assert.IsType<int>(result); Assert.IsNotType<string>(result); |
| IsAssignableFrom | Değerin tür türetmesini kontrol eder. | Assert.IsAssignableFrom<object>(result);  //ya da  var collection = new List<object>();  Assert.IsAssignableFrom<IEnumerable<object>>(collection); |
| Null/NotNull | Değerin null olup olmadığını kontrol eder. | Assert.Null(result); Assert.NotNull(result); |

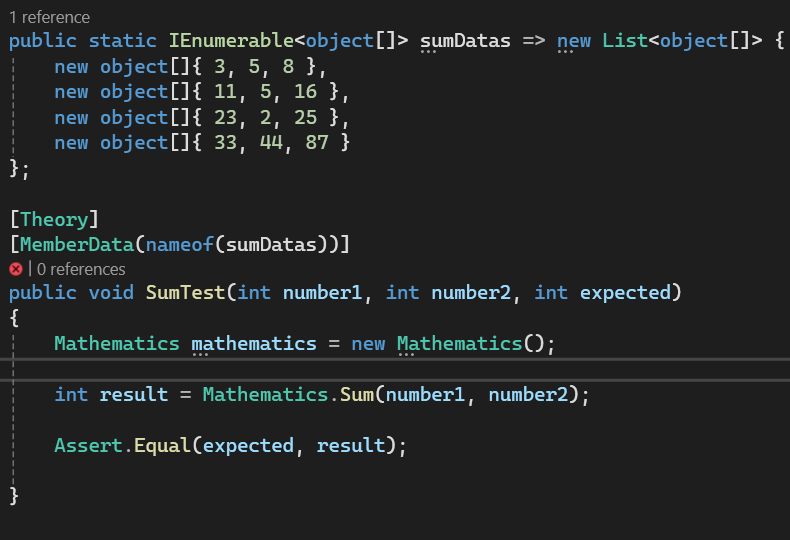
Test metodları parametre alıyorsa theory attribute’u ile işaretliyoruz. Ardından parametrelerini ‘InlineData’ attribute’u ile veriyoruz.





33 + 44 87 değil onun için testten geçemedim.

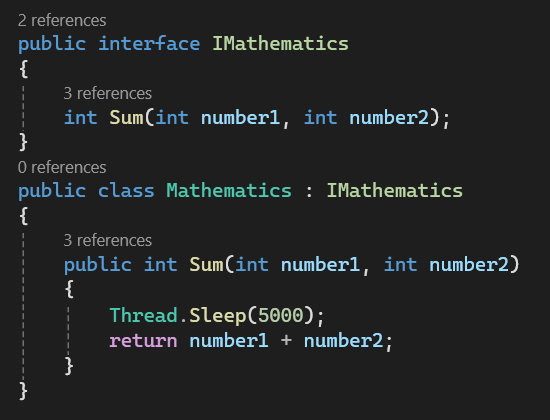
Test miktarı fazla ise memberdata attributeu ile böyle yazabiliriz.



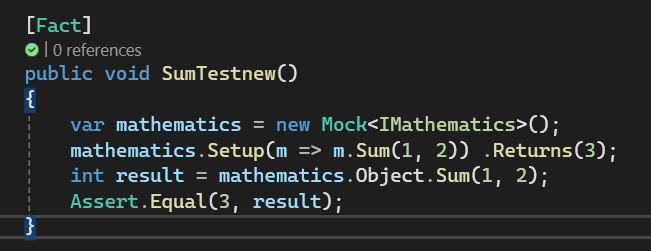
**Mock**

Test edilmek istenilen sınıfların gerçek nesnelerini kullanmak yerine onları simüle etmemizi sağlayan frameworktür.

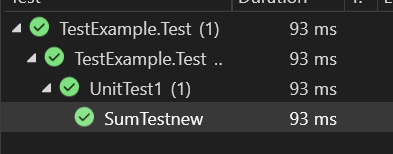
Mesela,  test edilecek bir sınıfın haddinden fazla uzun süreliğine çalışan metotları varsa ilgili sınıfları simüle etmek ve sadece davranışlarını modellemek testten geçip geçmediğine dair değerlendirme yapmak için yeterlidir.



Sum metodu 5 saniye sürdüğünü düşünürsek her test sürecinde beklememiz gerekir. Metodu taklit edersek test etme süremiz kısalır.



Mock’ sınıfına ‘IMathematics’ interface’i verilmekte ve böylece hangi interface içerisindeki metotların simüle edileceği bilir. Sonra  interface içerisinde simüle edilecek olan metot ‘Setup’ edilmekte ‘1’ ve ‘2’ değerlerine karşı ‘3’ değerinin dönmesi gerektiği bilir. Son olarakta ‘Mock’ nesnesi üzerinden ‘Object’ property’si ile üretilen nesne çağrılmakta ve ilgili metot tetiklenmektedir.



Mock üzerinden ayrı değerlendirmeler

**Verify**  
Bir metodun kaç kez çalıştığını test edebilmek için kullanılan metottur.

**Throws**  
Bir metodun fırlattığı exception’ı test edebilmemizi sağlayan metottur.

**It.IsAny<T>**

Opsiyonel parametre için