Bu dokümanda, C# dilindeki Nullable Değer Tipleri konusunu ele alıyoruz. Normalde boş (null) olamayan int, bool, DateTime gibi değer tiplerinin, ? operatörü yardımıyla nasıl null değer alabilir hale getirildiği anlatılmaktadır. Doküman, bu tiplerin tanımlanması, değerlerine güvenli bir şekilde erişilmesi ve modern C# operatörleriyle (??, ?.) verimli kullanımı gibi konuları, sade ve anlaşılır örneklerle açıklamaktadır. Bu çalışma, özellikle veritabanı veya opsiyonel kullanıcı verileriyle çalışırken karşılaşılan null durumlarını hatasız yönetmek ve daha sağlam kodlar yazmak için temel bir rehber niteliğindedir.

NULLABLE DEĞER TİPLERİ

Nullable operatörü, Null conditional ve Null coalescing

Oğuzhan Karagüzel

İÇİNDEKİLER

GİRİŞ

Bu dokümanda, C# programlama dilinin önemli bir özelliği olan **Nullable Değer Tipleri** konusunu detaylı bir şekilde inceliyoruz. Önceki derslerimizde int, double, bool gibi değer tiplerinin (value types) her zaman bir değere sahip olması gerektiğini, null (değersiz/boş) olamayacağını öğrenmiştik. Ancak, gerçek dünya senaryolarında bir verinin "yokluğunu" veya "belirsizliğini" ifade etmemiz gereken durumlarla sıkça karşılaşırız.

Örneğin, bir veritabanı tablosundaki "Doğum Tarihi" sütunu boş bırakılabilir olabilir veya bir kullanıcı web formundaki "Yaş" alanını doldurmayabilir. İşte bu gibi durumlarda, C#'ta değer tiplerine null atanabilme yeteneği kazandıran **Nullable tipler** devreye girer.

Bu doküman, Nullable<T> yapısının ne olduğunu, nasıl kullanıldığını, modern C# operatörleriyle nasıl daha verimli hale getirildiğini ve konunun mantığını tam olarak kavramanız için kendi Nullable yapımızı nasıl yazabileceğimizi adım adım anlatmak amacıyla hazırlanmıştır.

Ayrıca değinmek gerekir ki bu aşamaya kadar geldiğinize göre yavaş yavaş kod yazıyorsunuz. Bundan dolayı bu dokümanda, (Hatta önceki birkaç dokümanda da) kodlar doğrudan yazılmıştır. Bu sayede rahatlıkla kopyalayıp yapıştırabilirsiniz. Ancak bu durum alışkanlık haline gelmesin!

NULLABLE DEĞER TİPİ NEDİR?

Nullable Değer Tipi, normalde null değeri alamayan bir değer tipine (int, bool, DateTime, custom struct'lar vb.) null olabilme özelliği katan özel bir yapıdır.

C#'ta bu yapıyı kullanmanın iki yaygın yolu vardır. İkisi de temelde aynı anlama gelir:

1. **Soru İşareti ? Notasyonu (Kısa Yol):** En sık kullanılan ve en pratik yöntemdir. Bir değer tipinin sonuna ? ekleyerek onu nullable yapabilirsiniz.
2. **System.Nullable<T> Generic Struct'ı:** ? notasyonunun arka planda derleyici tarafından dönüştürüldüğü asıl yapıdır. T, int veya bool gibi herhangi bir değer tipini temsil eden generic bir parametredir.

```
// ---- Soru İşareti (?) Kullanımı ----


// Normal int null alamaz, bu satır derleme hatası verir.
// int normalSayi = null;


// Nullable int, hem bir sayı değeri hem de null alabilir.
int? nullableSayi = null;
nullableSayi = 100; // Geçerli.
nullableSayi = null; // Tekrar null atanabilir, geçerli.
```

```
// Diğer değer tipleri için örnekler
double? agirlik = null;
bool? cevapVerildiMi = null;
DateTime? bitisTarihi = null;


// ---- Nullable<T> Kullanımı (Yukarıdakiyle birebir aynı) ----
Nullable<int> digerNullableSayi = null;
Nullable<bool> digerCevap = true;


// İki yazım da aynı kapıya çıkar. C# programcıları genellikle '?'
kullanır.
// int? <==> Nullable<int>
```

Önemli Not: Nullable tipler sadece **değer tipleri (value types)** için geçerlidir. class, string, object gibi referans tipleri (reference types) zaten doğaları gereği null olabilirler, bu yüzden onlara ? eklemenin (C# 8.0 öncesi) bir anlamı yoktur.

NEDEN NULLABLE TİP KULLANIRIZ?

- **Veritabanı İşlemleri:** Veritabanı tablolarındaki sütunlar genellikle NULL (boş) değerlere izin verir. Bu sütunları koddaki değişkenlere eşlerken, bu "boşluk" durumunu null ile ifade etmek en doğal yoldur.
- **API ve Veri Transferi (DTO):** Bir API'den gelen JSON verisinde bazı alanlar opsiyonel olabilir. Bu alanları temsil eden C# sınıflarındaki karşılıkları Nullable yapmak, veri eksikliğinde programın çökmesini engeller.
- **Opsiyonel Fonksiyon Parametreleri:** Bir metoda gönderilen bir değerin "isteğe bağlı" olduğunu belirtmek için Nullable tipler kullanılabilir.

NULLABLE DEĞERLERE ERİŞİM

Nullable bir değişkene sahip olduğumuzda, içindeki değere doğrudan erişmeye çalışmak risklidir. Eğer değişken o anda null ise, program System.InvalidOperationException hatası vererek çöker.

```
int? puan = null;

// Console.WriteLine(puan.Value); // HATA! Nullable object must have a value.
```

Bu durumu güvenli bir şekilde yönetmek için Nullable<T> yapısının sunduğu üyeleri kullanırız.

HASVALUE ÖZELLİĞİ

Bir Nullable değişkenin içinde geçerli bir değer olup olmadığını kontrol eder. bool tipinde bir sonuç döner.

- Değişken null değilse true.
- Değişken null ise false.

VALUE ÖZELLİĞİ

HasValue true ise, değişkenin içerdiği asıl değeri verir. **Sadece ve sadece HasValue kontrolü yapıldıktan sonra kullanılmalıdır.**

```
int? kullaniciYasi = null;


// HasValue ile kontrol edelim
if (kullaniciYasi.HasValue)
{
    // Bu blok çalışmayacak çünkü kullaniciYasi null.
    int yas = kullaniciYasi.Value;
    Console.WriteLine($"Kullanıcının yaşı: {yas}");
}
else
{
    Console.WriteLine("Kullanıcı yaş bilgisini girmemiş.");
}
// Çıktı: Kullanıcı yaş bilgisini girmemiş.


// Şimdi değer atayalım
kullaniciYasi = 30;
if (kullaniciYasi.HasValue)
{
```

```
    int yas = kullaniciYasi.Value;
    Console.WriteLine($"Kullanıcının yaşı: {yas}");
}
// Çıktı: Kullanıcının yaşı: 30
```

Pratik Alternatif: if (degisken.HasValue) yerine if (degisken != null) kontrolünü de kullanabilirsiniz. Bu, kodun okunabilirliğini artırır.

GetValueOrDefault() METODU

Bu metot, if-else bloklarına gerek kalmadan güvenli bir şekilde değer almanın en pratik yollarından biridir.

- Değişken bir değere sahipse, o değeri döndürür.
- Değişken null ise, tipin varsayılan değerini döndürür (int için 0, bool için false vb.).

```
int? siparisAdedi = null;
int? stokKodu = 12345;

int adet = siparisAdedi.GetValueOrDefault(); // null olduğu için int'in varsayılanı olan 0 döner.
int kod = stokKodu.GetValueOrDefault();      // Değeri olduğu için 12345 döner.

Console.WriteLine($"Adet: {adet}");     // Çıktı: Adet: 0
Console.WriteLine($"Stok Kodu: {kod}"); // Çıktı: Stok Kodu: 12345
```

Ayrıca, null durumunda dönmesini istediğiniz özel bir varsayılan değeri de belirtebilirsiniz:

```
// Eğer siparisAdedi null ise, varsayılan olarak 1 dönsün.
int gosterilecekAdet = siparisAdedi.GetValueOrDefault(1);
Console.WriteLine($"Gösterilecek Adet: {gosterilecekAdet}"); // Çıktı: Gösterilecek Adet: 1
```

OPERATÖRLERLE ÇALIŞMA

Modern C# sözdizimi, Nullable tiplerle çalışmayı çok daha kolaylaştıran güçlü operatörler sunar.

NULL-COALESCING OPERATÖRÜ (??)

"Eğer sol taraftaki ifade null ise, sağ taraftaki değeri kullan; değilse, sol taraftakini kullan" anlamına gelen ikili bir operatördür. GetValueOrDefault(varsayilanDeger) metodunun kısa ve okunabilir halidir.

```
string kullaniciAdi = null;


// ?? operatörü ile: Eğer kullaniciAdi null ise "Misafir" ata.

string gosterilecekAd = kullaniciAdi ?? "Misafir";


Console.WriteLine($"Hoş Geldin, {gosterilecekAd}"); // Çıktı: Hoş Geldin,
Misafir


// Örnek 2:

int? urunFiyati = null;

int sepetFiyati = urunFiyati ?? 0; // Eğer fiyat belirtilmemişse 0 kabul
et.

Console.WriteLine($"Sepet Tutarı: {sepetFiyati} TL"); // Çıktı: Sepet
Tutarı: 0 TL
```

NULL-CONDITIONAL OPERATÖRÜ (?.)

Bir nesne veya struct null ise, üyelerine (metot, property) erişmeye çalışmak NullReferenceException veya InvalidOperationException hatasına neden olur. ?. operatörü, bu zincirleme çağrıyı güvenli hale getirir.

- Eğer operatörün solundaki ifade null ise, sağdaki üye çağrılmaz ve tüm ifadenin sonucu null olur.
- Eğer null değilse, üye çağrılır ve sonuç döndürülür.

```
// Genellikle referans tipleriyle kullanılır. (OOP'de daha derinlemesine
öğreneceksiniz):

string metin = null;

int? uzunluk = metin?.Length; // metin null olduğu için .Length'e
erişilmez, uzunluk null olur.

Console.WriteLine($"Metin Uzunluğu: {uzunluk}"); // Çıktı boş kalır (null).


metin = "C#";
```

```
uzunluk = metin?.Length; // metin null olmadığı için .Length çalışır, uzunluk 2 olur.

Console.WriteLine($"Metin Uzunluğu: {uzunluk}"); // Çıktı: Metin Uzunluğu: 2



// Nullable<T> için kullanımına örnek:

public struct Nokta { public int X; public int Y; }


Nokta? nokta = null;

int? xKoordinati = nokta?.X; // 'nokta' null olduğu için .X'e erişilmez. sonuç null olur. Hata vermez!

Console.WriteLine($"X Koordinatı: {xKoordinati}"); // Çıktı boş kalır.
```

Not: ?. operatörü kullanıldığında, sonuç tipi de otomatik olarak Nullable olur (int -> int?, bool -> bool?).

İLERİ SEVİYE: KENDİ MyNullable<T> STRUCT'IMIZI OLUŞTURMAK

Nullable<T>'nin nasıl çalıştığını tam olarak anlamak için, onun basitleştirilmiş bir versiyonunu kendimiz yazabiliriz. Bu, struct ve generic yapılar hakkındaki bilgimizi pekiştirirken konunun iç mantığını kavramamızı sağlar.

```
// Sadece değer tipleriyle (struct) çalışması için 'where T : struct' kısıtlaması ekliyoruz.

public struct MyNullable<T> where T : struct

{

    private readonly T _value;      // Gerçek değeri tutan alan

    private readonly bool _hasValue;  // Bir değerinin olup olmadığını belirten bayrak



    // Bu yapıcı metot, dışarıdan bir değer verildiğinde çalışır.

    public MyNullable(T value)

    {
```

```
        _value = value;
        _hasValue = true; // Artık bir değeri var!
    }

    // Değer olup olmadığını dışarıya bildiren özellik (property)
    public bool HasValue => _hasValue;

    // Gerçek değere erişim sağlayan özellik
    public T Value
    {
        get
        {
            if (!HasValue)
            {
                // Standart hatayı fırlatıyoruz
                throw new InvalidOperationException("Nullable nesnesi bir
değere sahip olmalıdır.");
            }
            return _value;
        }
    }

    public T GetValueOrDefault()
    {
        // Eğer değer varsa _value'yu, yoksa T'nin varsayılanını (default)
döndür.
        return _hasValue ? _value : default(T);
    }
}

// ---- Kullanım Örneği ----
// var benimSayim = new MyNullable<int>();
// Console.WriteLine(benimSayim.HasValue); // False
```

```
// var doluSayim = new MyNullable<int>(42);
// Console.WriteLine(doluSayim.HasValue); // True
// Console.WriteLine(doluSayim.Value);    // 42
```

Bu örnek, Nullable<T>'nin aslında içinde değeri ve değerin varlığını belirten bir bayrağı tutan akıllı bir struct'tan ibaret olduğunu gösterir.

HIZLI BAŞVURU TABLOSU

Kavram / Operatör	Açıklama	Örnek Kullanım
int? / Nullable<int>	Bir değer tipini "boş bırakılabilir" yapar.	int? sayi = null;
.HasValue	Değişkenin null olup olmadığını kontrol eder (true/false).	if (sayi.HasValue)
.Value	HasValue kontrolünden sonra, değişkenin içindeki asıl değeri alır.	int deger = sayi.Value;
.GetValueOrDefault()	Değer varsa onu, yoksa tipin varsayılanını (0, false) veya belirtilen bir değeri verir.	int deger = sayi.GetValueOrDefault(10);
?? (Null-Coalescing)	Sol taraf null ise sağdaki değeri, değilse soldakini alır.	int deger = sayi ?? 10;
?. (Null-Conditional)	Sol taraf null ise ifadeyi çalıştırmaz ve null döner; program çökmez.	int? x = nokta?.X;
== null	HasValue kontrolünün daha okunabilir bir alternatifidir.	if (sayi == null)

ÖZET VE SONRAKİ ADIMLAR

Bu derste öğrendiklerimizi özetleyecek olursak:

- **Struct'lar ve Değer Tipleri** normalde null olamaz.

- ? operatörü veya Nullable<T> struct'ı ile onlara null atanabilme özelliği kazandırdık.
- HasValue, Value ve GetValueOrDefault() gibi üyelerle bu değişkenlere **güvenli erişimin** önemini anladık.
- ?? ve ?. gibi modern operatörlerle kodumuzu nasıl daha **temiz ve hataya dayanıklı** yazabileceğimizi gördük.

İpuçları:

- **Okunabilirlik Önceliğiniz Olsun:** Karmaşık if-else blokları yerine ?? operatörünü kullanmak genellikle kodu daha anlaşılır kılar.
- **Aşırı Kullanımdan Kaçının:** Bir değişkenin "yokluğu" mantıksal olarak bir anlam ifade etmiyorsa, onu Nullable yapmayın. Nullable tipler, "opsiyonel" veriler için bir araçtır.

Nullable tipler, özellikle nesne yönelimli programlama (OOP) ve veritabanı odaklı uygulamalar geliştirdiğinizde sürekli olarak karşınıza çıkacak bir konsepttir. Bu temeli sağlam atmak, ileride karşılaşacağınız daha karmaşık problemlerin çözümünde size büyük avantaj sağlayacaktır.