Bu doküman, C# programlamada kod okunabilirliğini ve güvenliğini artıran enum (numaralandırma) yapısını tanıtmaktadır. 'Sihirli sayıların' (magic numbers) yarattığı karmaşayı ortadan kaldırarak temiz ve sürdürülebilir kod yazmanın temellerini keşfedin.

ENUM'LAR

Kodunuza Anlamlı Etiketler Vermek

Oğuzhan Karagüzel

İçindekiler

GİRİŞ

Hayatımızdaki birçok şeyi etiketlerle veya isimlerle daha kolay anlarız. Trafik ışıklarındaki renklerin "Dur", "Hazırlan" ve "Geç" gibi anlamları vardır; sadece "Renk 1", "Renk 2" demeyiz. Bu isimler sayesinde ne yapacağımızı hemen anlarız.

İşte programlamadaki enum yapısı da tam olarak bu işe yarar. Sayılar veya belirsiz karakterler yerine, programımızdaki belirli durumlara ve seçeneklere net, anlaşılır ve akılda kalıcı etiketler vermemizi sağlar. 1 yerine Aktif, 2 yerine Pasif demek gibi...

Bu dersimizde, kodumuza bu güçlü etiketleme sistemini nasıl dahil edeceğimizi, yani enum'ları nasıl kullanacağımızı öğreneceğiz. Bu sayede yazdığınız kodlar sadece bilgisayar için değil, aynı zamanda kodu okuyan diğer insanlar (ve gelecekteki kendiniz!) için de çok daha anlamlı hale gelecek

C# PROGRAMLAMADA ENUM (NUMARALANDIRMA) KAVRAMI

Bu dokümanda C# programlama dilinin en kullanışlı ve kod okunabilirliğini artıran özelliklerinden biri olan enum yani numaralandırma yapısını inceleyeceğiz. Temel bir C# bilgisine sahip olduğunuzu varsayarak, enum'ların ne olduğunu, neden kullanıldığını ve kodlarımızı nasıl daha profesyonel hale getirdiğini adım adım öğreneceğiz.

Sorun: "Sihirli Sayılar" (Magic Numbers)

Enum konusuna girmeden önce, onların çözdüğü bir problemi anlamamız gerekiyor: sihirli sayılar.

Diyelim ki bir oyun geliştiriyorsunuz ve oyunun üç farklı zorluk seviyesi var: Kolay, Orta ve Zor. Bu seviyeleri kodda temsil etmek için aklınıza ilk gelen yöntem sayılar kullanmak olabilir.

```
// 1 = Kolay

// 2 = Orta

// 3 = Zor

int mevcutSeviye = 1;

if (mevcutSeviye == 1)

{

    Console.WriteLine("Oyun kolay modda başlatıldı.");

}

else if (mevcutSeviye == 2)

{

    Console.WriteLine("Oyun orta modda başlatıldı.");

}
```

Bu kod çalışır, ancak birkaç ciddi sorunu vardır:

Anlaşılırlık Zayıf: Birkaç ay sonra bu koda baktığınızda mevcutSeviye == 1 ne anlama geliyordu? "1" sayısının "Kolay" seviyesini temsil ettiğini hatırlamak için kodun başındaki yorumları bulmanız gerekir. Buna "sihirli sayı" denir çünkü anlamı kodun içinde gizlidir.

Hata Yapmaya Açık: Yanlışlıkla mevcutSeviye = 5 gibi geçersiz bir değer atarsanız, programınızda mantık hataları oluşur ve bunu fark etmek zor olabilir.

Bakımı Zor: Yeni bir "Çok Zor" seviyesi (örneğin 4) eklemek isterseniz, kodunuzdaki 1, 2, 3 sayılarını kullanan her yeri bulup güncellemeniz gerekir.

İşte bu noktada enum'lar devreye girerek bu sorunları şık bir şekilde çözer.

ENUM NEDİR?

Enum (enumeration kelimesinin kısaltması), bir grup sabit değere anlamlı isimler vermemizi sağlayan özel bir veri türüdür. Az önceki sayıları, anlamlı kelimelerle değiştirmemize olanak tanır.

Temel Söz Dizimi (Syntax):

Bir enum tanımlamak son derece basittir.

```
enum ZorlukSeviyesi
{
    Kolay,
    Orta,
    Zor
}
```

Bu kadar! Artık ZorlukSeviyesi adında yeni bir veri türümüz var. Bu tür sadece Kolay, Orta ve Zor değerlerini alabilir.

ENUM NASIL KULLANILIR?

Şimdi sihirli sayılarla yazdığımız kodu yeni enum'ımızla tekrar yazalım:

```
// Bir enum değişkeni tanımlama ve değer atama
ZorlukSeviyesi mevcutSeviye = ZorlukSeviyesi.Orta;

// switch-case ile kontrol etmek çok daha okunabilir ve güvenlidir
switch (mevcutSeviye)
{
    case ZorlukSeviyesi.Kolay:
        Console.WriteLine("Oyun kolay modda başlatıldı.");
        break;
    case ZorlukSeviyesi.Orta:
```

```
        Console.WriteLine("Oyun orta modda başlatıldı.");
        break;
    case ZorlukSeviyesi.Zor:
        Console.WriteLine("Oyun zor modda başlatıldı.");
        break;
    default:
        Console.WriteLine("Bilinmeyen bir seviye seçildi.");
        break;
}
```

Ne kazandık?

*   **Okunabilirlik:** `mevcutSeviye == ZorlukSeviyesi.Orta` ifadesi, `mevcutSeviye == 2` ifadesinden kat kat daha anlaşılırdır. Kod kendini belgeliyor.
*   **Tür Güvenliği (Type Safety):** `mevcutSeviye` değişkenine `ZorlukSeviyesi` enum'ı dışında bir değer atayamazsınız. Örneğin `mevcutSeviye = 5;` yazmak derleme hatası verir. Bu, programımızı daha en başından hatalara karşı korur.
*   **IntelliSense Desteği:** Visual Studio gibi bir geliştirme ortamında `ZorlukSeviyesi.` yazdığınız anda, `Kolay`, `Orta`, `Zor` seçenekleri otomatik olarak karşınıza çıkar. Bu hem yazım hatalarını önler hem de kod yazmayı hızlandırır.

ENUM'LARIN "PERDE ARKASI": SAYISAL DEĞERLER

Aslında enum'lar, derleyici için hala sayılardır. Varsayılan olarak, C# bir enum'ın ilk elemanına `0` değerini, ikinciye `1`, üçüncüye `2` ve bu şekilde devam ederek atar.

```
enum ZorlukSeviyesi
{
    Kolay,  // Değeri 0
    Orta,   // Değeri 1
    Zor     // Değeri 2
}
```

Bu sayısal değere ihtiyacınız olursa, tür dönüştürme (casting) yapabilirsiniz:

```
int ortaSeviyeDegeri = (int)ZorlukSeviyesi.Orta;

Console.WriteLine(ortaSeviyeDegeri); // Ekrana "1" yazar
```

Sayısal Değerleri Özelleştirme:

Bazen varsayılan 0, 1, 2... sıralaması yerine kendi sayısal değerlerinizi atamak isteyebilirsiniz. Örneğin, bir API ile çalışıyorsunuz ve o API sizden belirli sayısal kodları istiyor olabilir.

```
enum IslemDurumu

{

    Basarili = 200,

    Bulunamadi = 404,

    SunucuHatasi = 500

}


IslemDurumu durum = IslemDurumu.Basarili;

int durumKodu = (int)durum; // durumKodu'nun değeri 200 olur
```

Değer atamaya herhangi bir yerden de başlayabilirsiniz. Geri kalanı otomatik olarak artacaktır.

```
enum Siralama

{

    Birinci = 1,

    Ikinci,      // Otomatik olarak 2 olur

    Ucuncu       // Otomatik olarak 3 olur

}
```

ENUM'LARIN ALTINDAKİ VERİ TÜRÜNÜ DEĞİŞTİRME

Varsayılan olarak her enum, int (32-bit integer) veri türünü temel alır. Ancak bellekten tasarruf etmek veya daha büyük sayılara ihtiyaç duymak gibi durumlarda bu temel türü değiştirebilirsiniz (byte, sbyte, short, ushort, int, uint, long, ulong olabilir). Özellikle byte, çok sayıda enum değeriniz yoksa bellek optimizasyonu için iyi bir seçenektir.

```
// Bu enum, her bir değer için 4 byte yerine sadece 1 byte yer kaplar.

enum CokKucukAyar : byte

{

    Kapali,

    Acik,

    Otomatik

}
```

PRATİK ENUM METOTLARI

.NET, enum'larla çalışmayı kolaylaştıran bazı yerleşik metotlar sunar.

ToString(): Enum değerinin ismini string olarak verir.

```
ZorlukSeviyesi seviye = ZorlukSeviyesi.Zor;

string seviyeAdi = seviye.ToString(); // seviyeAdi = "Zor"
```

Enum.Parse(): String bir ifadeyi enum değerine çevirir. Kullanıcıdan metin olarak giriş aldığınızda çok kullanışlıdır.

```
string kullaniciGirdisi = "Orta";

ZorlukSeviyesi secilenSeviye =
(ZorlukSeviyesi)Enum.Parse(typeof(ZorlukSeviyesi), kullaniciGirdisi);


Console.WriteLine(secilenSeviye); // Ekrana "Orta" yazar
```

Enum.GetValues(): Bir enum'ın tüm olası değerlerini içeren bir dizi döndürür. Örneğin bir arayüzdeki "dropdown" menüsünü doldurmak için harikadır.

```
foreach (var seviye in Enum.GetValues(typeof(ZorlukSeviyesi)))

{

    Console.WriteLine($"Seçenek: {seviye} (Değeri: {(int)seviye})");

}

// Çıktı:

// Seçenek: Kolay (Değeri: 0)

// Seçenek: Orta (Değeri: 1)

// Seçenek: Zor (Değeri: 2)
```

İLERİ SEVİYE KONU: [FLAGS] ENUMLARI

Bazen bir değişkenin, birden fazla seçeneği aynı anda temsil etmesini isteyebilirsiniz. Örneğin, bir kullanıcının dosya izinleri: Okuma, Yazma, Çalıştırma. Bir kullanıcı hem okuma hem de yazma iznine sahip olabilir.

Bunu başarmak için [Flags] attribute'ünü ve 2'nin katları şeklinde giden değerleri kullanırız.

```
[Flags] // Bu attribute, enum'ın bit alanı olarak kullanılabileceğini
belirtir.

enum Izinler

{

    Yok = 0,        // 0000

    Okuma = 1,      // 0001

    Yazma = 2,      // 0010
```

```
    Calistirma = 4  // 0100
}

// Bir kullanıcıya hem okuma hem de yazma izni verelim
// "|" (bitwise OR) operatörü kullanılır.
Izinler kullaniciIzinleri = Izinler.Okuma | Izinler.Yazma;

Console.WriteLine(kullaniciIzinleri); // Çıktı: Okuma, Yazma

// Kullanıcının belirli bir izni var mı diye kontrol edelim
// "&" (bitwise AND) operatörü kullanılır.
if ((kullaniciIzinleri & Izinler.Okuma) == Izinler.Okuma)
{
    Console.WriteLine("Kullanıcının Okuma izni var.");
}

if ((kullaniciIzinleri & Izinler.Yazma) == Izinler.Yazma)
{
    Console.WriteLine("Kullanıcının Yazma izni var.");
}

if ((kullaniciIzinleri & Izinler.Calistirma) == Izinler.Calistirma)
{
    Console.WriteLine("Kullanıcının Çalıştırma izni var.");
}
else
{
    Console.WriteLine("Kullanıcının Çalıştırma izni YOK.");
}
```

Bu konu biraz daha ileri seviye olsa da, enum'ların ne kadar esnek olabileceğini göstermesi açısından önemlidir.

ÖZET: NEDEN ENUM KULLANMALIYIZ?

Kodun Okunabilirliğini Artırır: ZorlukSeviyesi.Kolay demek 1 demekten çok daha anlamlıdır.

Hataları Önler: Sadece tanımlı değerleri kullanmanıza izin vererek yanlış değer atamalarının önüne geçer (Tür Güvenliği).

Bakımı Kolaylaştırır: Yeni bir seviye eklemek veya bir değeri değiştirmek istediğinizde sadece enum tanımını güncellemeniz yeterlidir.

Geliştirme Sürecini Hızlandırır: Kod tamamlama (IntelliSense) özellikleri sayesinde olası değerleri anında görerek daha hızlı ve hatasız kod yazarsınız.

Enum'lar, basit gibi görünmelerine rağmen C#'ta temiz, sürdürülebilir ve profesyonel kod yazmanın temel taşlarından biridir. Onları kullanmayı alışkanlık haline getirdiğinizde, kodlarınızın kalitesinde belirgin bir artış göreceksiniz.