
Records
Object Oriented Programming

# Record'ı Anlayabilmek İçin Ön Hazırlık (Init-Only Properties)

C# 9.0'da, herhangi bir nesnenin propertylerine ilk değerlerinin verilmesi ve sonraki süreçte bu değerlerin değiştirilmemesini garanti altına almamızı sağlayan **Init – Only Properties** Özelliği gelmiştir.

Bu özellik sayesinde nesnenin sadece ilk yaratılış anında propertylerine değer atanmakta ve böylece iş kuralları gereği runtime'da değeri değişmemesi gereken nesneler için ideal bir önlem alınmaktadır.

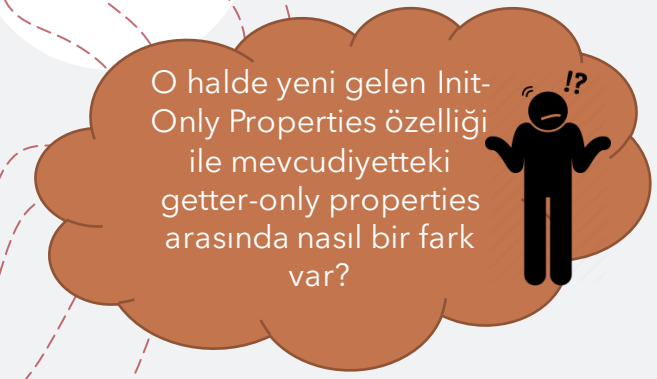
*Init-Only properties, developer açısından süreç esnasında değiştirilmemesi gereken property değerlerinin -yanlışlıkla- değiştirilmesinin önüne geçmekte ve böylece olası hata ve bug'lardan yazılımı arındırmaktadır.*

Property'nin değerinin değiştirilmemesi mi?

İyi de bunun için zaten '**getter-only-properties**'ler yok mu?

```
class Book
{
    public string Name { get; } = "Kutsal İsyan";
    public string Author { get; }
    public Book()
    {
        Author = "Hasan İzzet Dinamo";
    }
}
```

**Evet, haklısınız...!**
**Yandaki görselde olduğu gibi tanımlanan propertyler sadece get oldukla rından dolayı ya tanımlandıkları anda ya da sadece constructor'dan değer alabilmektedirler.**




Bu ikisi arasındaki temel fark esasında **Object Initializer** işlevselliğinden kaynaklanmaktadır.

```
Book book = new Book
{
    Author = "Kutsal İsyan",
    Name = "Hasan İzzet Dinamo"
};
```

Object Initializer özelliğinden getter-only-properties özelliği ile istifade edememekteyiz, lakin C# 9.0 ile gelen init-only-properties özelliği ile kullanabilmekteyiz.

İşte her iki özellik arasındaki temel fark buradadır...

```
Book book = new Book
{
    Author = "Kutsal İsyan",
    Name = "Hasan İzzet Dinamo"
};

book.Name = "Sabuncuoğlu Şerafettin";
```

```
class Book
{
    public string Name { get; init; } = "Kutsal İsyan";
    public string Author { get; init; }
    public Book()
    {
        Author = "Hasan İzzet Dinamo";
    }
}
```

init, get keyword'ü olmaksızın kullanılamaz.
Ayrıca yapısı gereği bu semantikte set bloğuda kullanılamaz!

get; init; konsepti bir araya geldiği vakit bu property readonly olmakta ve ilk değerlerini constructor yahut auto property initializers üzerinden alabileceği gibi 'init' sayesinde object initializer'dan da alabilmektedir.

# Properties Tanımlama

Init-Only Properties özelliğine 'init' keyword'ü eşlik etmektedir.

Ayrıca getter-only-properties ile çalışmaktansa readonly bir field üzerinde işlem yapmamız gerekiyorsa eğer aşağıdaki gibi 'init' bizlere eşlik edebilmektedir.

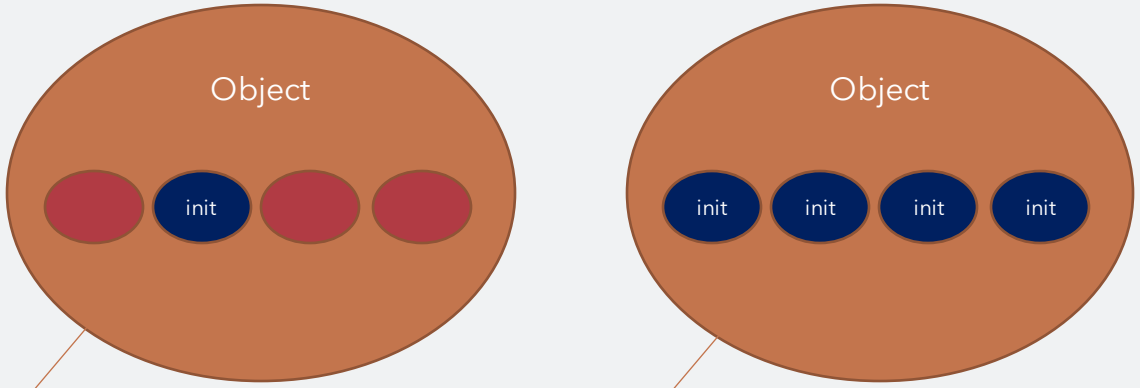
```
class Book
{
    private readonly string name;
    private readonly string author;
    public string Name
    {
        get => name;
        init => name = value;
    }
    public string Author
    {
        get => author;
        init => author = value;
    }
}
```

Böylece C# 9.0 yeniliklerinden olan init-only-properties sayesinde, nesnelerin ilk yaratılış esnasında salt okunabilir değerlerini vermek için Object Initializer'ı bloklamadan kullanabilmekteyiz.

Peki hoca! Tüm bunların Records ile ne ilgisi var! Records ne looo!

# Records Nedir?

- C# 9.0 ile gelen Init-Only Properties özelliği, nesne üretim esnasının dışında değişmez değerler oluşturulması için constructor ve auto property initializers yapısının yanında object initializer yapısının kullanılabilir olmasını sağlıyordu.




**Eğer ki tek bir property'de Sabitlik/Değişmemezlik/Salt Okunurluk/Readonlylik/Sadece Okunabilirlik amaç ediniliyorsa Init-Only Properties özelliği kullanılır.**

**Eğer ki bir objeyi bütünsel olarak değişmez yapmak istiyorsak o zaman daha fazlasına ihtiyacımız olacaktır. İşte bu ihtiyaca istinaden Records türü geliştirilmiştir.**

# Record, bir objenin topyekün olarak sabit/değişmez olarak kalmasını sağlamakta ve bu durumu güvence altına almaktadır.

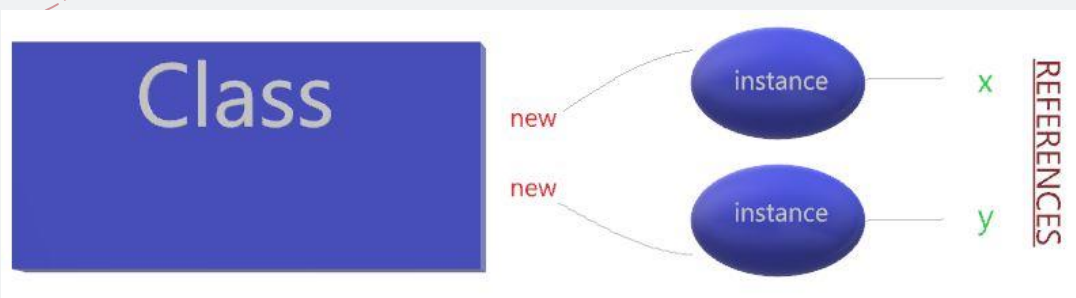
Böylece bu obje, artık değeri değişmeyeceğinden dolayı esasında objeden ziyade bir değer gözüyle bakılan bir yapıya dönüşmektedir.

Buradan yola çıkarak record'ları, içerisinde data barındıran lightweight( hafif) class'lar olarak değerlendirebiliriz.

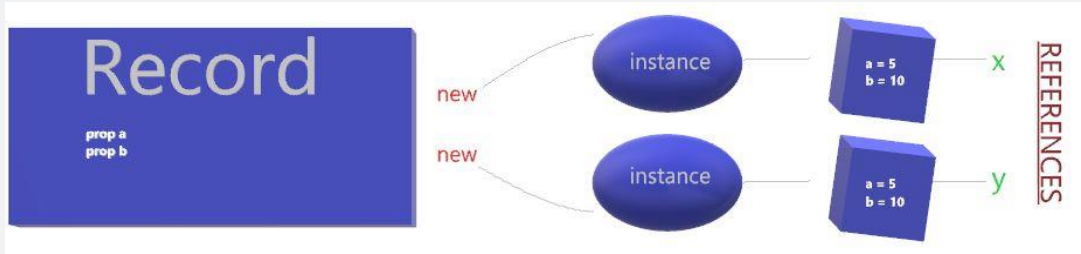
Record'lar, class'lara istinaden objeden ziyade içerisinde bulunan dataları sabitleyerek, nesneden ziyaden verilerini/datalarını öne çıkarmaktadır.

Record'lar bir class'tır. Sadece nesnelerinden ziyade, değerleri ön plana çıkmış bir class.

+ Class'lar da verisel olarak nesne ön plandadır ve bir farklı referansa sahip olan nesne farklı değer olarak algılanmaktadır.
+ Dolayısıyla Equals(x, y) karşılaştırması yanlıştır.




+ Record'lar ise verisel olarak değeri ön planda tutmaktadır. Sadece nesnel olarak bu veriler bir objede tutulmakta lakin değiştirilmemektedir.
+ Haliyle farklı objelerde de olsa, veriler(property değerleri) aynı olduğu sürece Equals(x, y) önermesi doğru olacaktır.




Her iki türde de veriler objede tutulmakta lakin record'lar class'lara nazaran, bir objeden ziyade topekün veri imajında olacak şekilde spesifik bir davraış sergilemektedirler.

# Record Tanımlama

Ayrıca record bildiğiniz class fıtratında bir yapılanma olduğu için içerisine her türlü class member'ları tanımlanabilmektedir. Haliyle tanımlanan propertylerin hepsinin init ile işaretlenmesi Record'ın esas amacına eşlik edecektir.

| Prototip | Örnek |
| --- | --- |
| `record [Name] { }` | `public record Book { public string Name { get; init; } public string Author { get; init; } }` |

Bu şekilde tanımlanan record'lara **Norminal Records** denmektedir.

# Record İle Class Arasındaki Fark Kritiği Yapalım!

+ Record'lar değiştirilemez objeler oluşturmamızı sağlamaktadır demiştik! Peki bu değiştirilemez objeleri class'lar ile gerçekleştiremez miyiz?

```
public class Employee
{
    public string Name { get; init; }
    public string Surname { get; init; }
    public int Position { get; init; }
}
```

+ Görüldüğü üzere içinde init olan(yani sadece okunabilir olan) propertyler barındıran bir class tasarlayabilir

ve

```
Employee employee1 = new Employee
{
    Name = "Gençay",
    Surname = "Yıldız",
    Position = 1
};
```

+ Bu sınıftan nesne oluştururken object initializer'da ilgili propertylere değerlerini set edebiliriz.

Yani anlayacağınız record'lar da ki değiştirilemez obje ihtiyacını kolaylıkla class ile karşılayabilmekteyiz!

**LAKİN!**

Bu nesnenin süreçte herhangi bir property değerini değiştirmek istediğimizde bunu gerçekleştirebilmek için yeni bir Employee nesnesi üretmemiz ve değişikliğin yapılacağı property dışında diğer property'leri bu nesneden eşleştirmemiz gerekecektir.

# Şöyle ki;

```
Employee employee1 = new Employee
{
    Name = "Ge
    Surname =
    Positio
};

Employee emp
{
    Name = employee1.Name,
    Surname = employee1.Surname,
    Position = 2
};
```

Halbuki bu senaryoda record kullansaydık, bu tarz kopyalama durumlarında with Expressions'dan faydalanabilmekte ve istenilen özellik hızlıca değiştirilebilmektedir.

Bu tarz bir senaryoda ne kadar çok property o kadar zor ve maliyetli kod demektir. Elbette reflection veya serialization ile kopyalama mantıkları uygulanabilir yahut bir auto mapping kütüphanesi kullanılabilir. Ama görüldüğü üzere bu tarz senaryolarda class'lar la çalışmak kaçınılmaz olarak her daim maliyetlidir.

**Bu yöntemin, adil sayıdan fazla property'e sahip olunduğu durumlarda can sıkıcı hale geleceği aşikar olsa gerek!**

# With Expressions

- Immutable türlerde çalışırken nesne üzeirnde değişiklik yapabilmek için ilgili nesneyi ya çoğaltmamız/klonlamamız(deep copy) ve üzerinde değişiklik yapmamız gerekmekte ya da manuel yeni bir nesne üretip mevcut nesnedeki değerleri, değişikliği yansıtılacak şekilde aktarmamız gerekmektedir.
- Misal, yan tarafta bu tarz durumlara istinaden yazılımcıların yılların deneyiminden getirdiği With function çözümü ele alınmaktadır.

```
public class Employee
{
    public string Name { get; init; }
    public string Surname { get; init; }
    public int? Position { get; init; }

    public Employee With(int position)
    {
        return new Employee
        {
            Name = this.Name,
            Surname = this.Surname,
            Position = position
        };
    }
}
```

```
Employee employee1 = new Employee
{
    Name = "Gençay",
    Surname = "Yıldız",
    Position = 1
};

Employee employee2 = employee1.With(2);
```

# With Expressions

Ya da aşağıdaki gibi record oluşturabilir ve With Function yazmadan direkt olarak with expressions'ları kullanabilirsiniz.

```
public record Employee
{
    public string Name { get; init; }
    public string Surname { get; init; }
    public int? Position { get; init; }
}
```

```
Employee employee1 = new Employee
{
    Name = "Gençay",
    Surname = "Yıldız",
    Position = 1
};

Employee employee2 = employee1 with { Position = 2 };
Employee employee3 = employee1 with { Name = "Hilmi", Position
Employee employee4 = employee2 with { Name = "Rıfkı", Position
```