

Bu bölümde, metot kullanımında ileri seviye teknikler olan Metot Overloading, Recursive Metotlar, Local Functions ve Expression-Bodied Metotları inceleyeceğiz. Bu yöntemler, kodlarınızı daha okunabilir ve etkili hale getirerek programlama becerilerinizi bir adım ileri taşımanıza yardımcı olacaktır.

Metodlar 4. Bölüm

Recursive, overload, expression bodied ve local metodlar.

Oğuzhan Karagüzel

İÇİNDEKİLER

ŞEKİLLER DİZİNİ.....	2
GİRİŞ	3
METODLAR 4. BÖLÜM	3
1. METOT OVERLOADING (AŞIRI YÜKLEME)	3
Metot Overloading Nedir?	3
Metot Overloading Kuralları.....	3
Metot Overloading Örneği	4
RECURSIVE (ÖZYİNELEMELİ) METOTLAR	5
Recursive Metot Nedir?.....	5
Recursive Metot Kullanım Kuralları.....	5
Recursive Metot Örneği - Sayıları Toplama	5
LOCAL FUNCTIONS (YEREL METOTLAR).....	7
Local Functions Nedir?	7
Local Functions Örneği	7
EXPRESSION-BODIED METOTLAR	8
Nedir?	8
Neden Kullanılır?	8
Nasıl Yazılır?.....	8
TEST	10
CEVAP ANAHTARI	12

ŞEKİLLER DİZİNİ

Şekil 1 Metot Overloading Örneği	4
Şekil 2 Metot Overloading Örneği Çıktısı.....	4
Şekil 3 Recursive Metot Örneği	5
Şekil 4 Recursive Metot Örneği Çıktısı	6
Şekil 5 Local Function Örneği.....	7
Şekil 6 Local Funtion Örneği Çıktısı.....	8
Şekil 7 Expression Bodied Örneği	9

Öğuzhan KARAGÜZEL

GİRİŞ

Modern yazılım geliştirme süreçlerinde, kodun okunabilirliği, esnekliği ve yeniden kullanılabilirliği kritik öneme sahiptir. Bu bağlamda, C# programlama dilinde metotlar, kodu düzenli, modüler ve anlaşılır hale getiren temel yapı taşları olarak öne çıkar. Bu döküman, metotların daha ileri düzey kullanım tekniklerini ele alarak, öğrencilerin güncel programlama yaklaşımlarını benimsemelerine yardımcı olmayı amaçlamaktadır.

Bu derste, aşağıdaki temel konulara odaklanılacaktır:

- **Metot Overloading (Aşırı Yükleme):** Aynı isimde tanımlanan metotların, farklı parametre tipleri veya sayılarına göre nasıl çalıştığı, tanımlama kuralları ve örnek uygulamaları.
- **Recursive (Özyinelemeli) Metotlar:** Kendi kendini çağıran metotların temel prensipleri, çıkış (base case) koşullarının önemi ve adım adım işlem mantığının örneklerle açıklanması.
- **Local Functions (Yerel Metotlar):** Bir metodun içinde tanımlanan alt metotların, kodun okunabilirliğini ve düzenliliğini nasıl artırdığı, kullanım avantajları.
- **Expression-Bodied Metotlar:** Daha kısa ve öz metot tanımlarının modern C# yazım teknikleriyle nasıl gerçekleştirilebileceği.

Bu teknikler, geliştiricilere kodlarını daha etkin, verimli ve bakımı kolay bir şekilde yazma imkanı sunar. Ayrıca, uygulamalı örnekler üzerinden konuların adım adım incelenmesi, öğrencilerin teorik bilgileri pratiğe dökerek öğrenmelerini sağlayacaktır.

METODLAR 4. BÖLÜM

1. METOT OVERLOADING (AŞIRI YÜKLEME)

Metot Overloading Nedir?

Metot aşırı yükleme, aynı isimde birden fazla metot tanımlayarak, farklı parametre türleri veya farklı sayıda parametrelerle çağrılabilmesini sağlamaktır.

Metot Overloading Kuralları

- Aynı isimde metotlar tanımlanabilir.
- Metotların **parametre sayısı veya türü farklı** olmalıdır.
- Geri dönüş tipi (return type) overloading için tek başına yeterli değildir.

Metot Overloading Örneği

```
class Program
{
    static void Yazdir(int sayi)
    {
        Console.WriteLine("Sayı: " + sayi);
    }

    static void Yazdir(double sayi)
    {
        Console.WriteLine("Ondalık Sayı: " + sayi);
    }

    static void Yazdir(string metin)
    {
        Console.WriteLine("Metin: " + metin);
    }

    static void Main()
    {
        Yazdir(10);
        Yazdir(10.5);
        Yazdir("Merhaba Dünya");
    }
}
```

Şekil 1 Metot Overloading Örneği

Şekil 2 Metot Overloading Örneği Çıktısı

Burada, **üç farklı Yazdir metodu** aynı isme sahip olmasına rağmen farklı parametre türlerini kabul ederek overloading işlemi gerçekleştirmiştir.

RECURSIVE (ÖZYİNELEMELİ) METOTLAR

Recursive Metot Nedir?

Recursive (özyinelemeli) metotlar, **kendi kendini çağırarak** metotlardır. Özellikle **sayıları toplama gibi işlemler** için kullanışlıdır.

Recursive Metot Kullanım Kuralları

- Her çağrıdan sonra **çıkış (base case) koşulu** belirlenmelidir. Aksi takdirde **sonsuz döngü** oluşur.
- Hafızayı aşırı kullanmamak için **verimli yazılmalıdır**.

Recursive Metot Örneği - Sayıları Toplama

Bu metot, n değerine kadar olan sayıların toplamını hesaplar. Özyinelemeli (recursive) olduğu için, **kendini çağırarak** işlemi gerçekleştirir.

```
class Program
{
    static int Toplam(int n)
    {
        Console.WriteLine($"Toplam({n}) çağrıldı"); // Adım adım takibi kolaylaştırmak için

        if (n == 0)
        {
            Console.WriteLine("Çıkış koşulu sağlandı: n == 0");
            return 0;
        }

        int sonuc = n + Toplam(n - 1);
        Console.WriteLine($"Toplam({n}) = {sonuc}");
        return sonuc;
    }

    static void Main()
    {
        Console.WriteLine("Toplam(5) işlemi başlatılıyor...");
        Console.WriteLine("Sonuç: " + Toplam(5));
    }
}
```

Şekil 3 Recursive Metot Örneği

```
Microsoft Visual Studio Debu x + v
Toplam(5) işlemi başlatılıyor...
Toplam(5) çağrıldı
Toplam(4) çağrıldı
Toplam(3) çağrıldı
Toplam(2) çağrıldı
Toplam(1) çağrıldı
Toplam(0) çağrıldı
Çıkış koşulu sağlandı: n == 0
Toplam(1) = 1
Toplam(2) = 3
Toplam(3) = 6
Toplam(4) = 10
Toplam(5) = 15
Sonuç: 15

C:\Users\Oguzh\OneDrive\Masaüstü\ConsoleApp1\ConsoleApp1\bin\Release\net9.0\ConsoleApp1.exe (process 14016) exited with
code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .|
```

Şekil 4 Recursive Metot Örneği Çıktısı

Adım Adım Çalışma Mantığı:

1. Toplam(5) çağrılır, 5 + Toplam(4) işlemi yapılır.
2. Toplam(4) çağrılır, 4 + Toplam(3) işlemi yapılır.
3. Toplam(3) çağrılır, 3 + Toplam(2) işlemi yapılır.
4. Toplam(2) çağrılır, 2 + Toplam(1) işlemi yapılır.
5. Toplam(1) çağrılır, 1 + Toplam(0) işlemi yapılır.
6. Toplam(0) çağrılır ve **çıkış koşulu sağlandığı için 0 döner**.
7. Geri dönüş işlemleri başlar ve hesaplanan değerler yukarıya iletilir.

Çıktı:

```
Toplam(5) işlemi başlatılıyor...
Toplam(5) çağrıldı
Toplam(4) çağrıldı
Toplam(3) çağrıldı
Toplam(2) çağrıldı
Toplam(1) çağrıldı
Toplam(0) çağrıldı
Çıkış koşulu sağlandı: n == 0
Toplam(1) = 1
Toplam(2) = 3
```

Toplam(3) = 6

Toplam(4) = 10

Toplam(5) = 15

Sonuç: 15

Bu şekilde, recursive fonksiyonların çalışma mantığını öğrencilerin **adım adım takip etmesi** sağlanarak daha iyi anlaşılması hedeflenmiştir.

LOCAL FUNCTIONS (YEREL METOTLAR)

Local Functions Nedir?

C# 7.0 ile gelen **yerel metotlar**, bir metodun içinde tanımlanan metotlardır. Bu, kodun **daha düzenli ve okunabilir** olmasını sağlar.

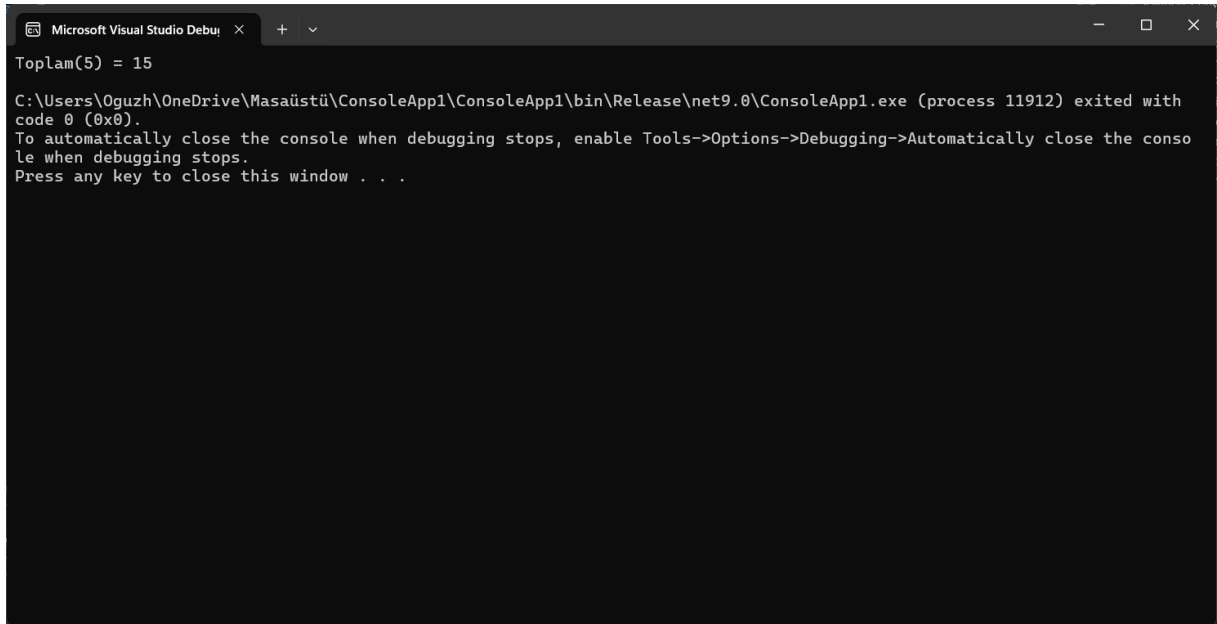
Recursive metotları local olarak çalıştırırsak;

Local Functions Örneği

```
class Program
{
    static void Main()
    {
        int Toplam(int n)
        {
            if (n == 0) return 0;
            return n + Toplam(n - 1);
        }

        Console.WriteLine("Toplam(5) = " + Toplam(5));
    }
}
```

Şekil 5 Local Function Örneği



```
Microsoft Visual Studio Debu x + v
Toplam(5) = 15
C:\Users\Oguzh\OneDrive\Masaüstü\ConsoleApp1\ConsoleApp1\bin\Release\net9.0\ConsoleApp1.exe (process 11912) exited with
code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Şekil 6 Local Funtion Örneği Çıktısı

Eğer metodunuzun içerisinde tekrarlayan kodlar var ise bunları daha okunabilir hale getirmek için kullanılabilecek harika yapılardır.

EXPRESSION-BODIED METOTLAR

Nedir?

C# dilinde, normalde metotlar süslü parantezler { } içinde yazılan kod bloklarına sahiptir. Ancak bazı metotlar, sadece tek bir ifade (yani tek satırlık bir hesaplama veya dönüş işlemi) yapıyorsa, bu kodu daha kısa ve okunabilir hale getirmek için *expression-bodied metotlar* kullanılır. Burada metot gövdesi, klasik süslü parantezler yerine => (ok) operatörü ile tek bir ifade şeklinde yazılır.

Neden Kullanılır?

- **Kısaltılmış Yazım:** Uzun yazım yerine, kısa ve net bir şekilde metodun ne yaptığını belirtir.
- **Okunabilirlik:** Özellikle basit dönüş değerleri veya tek satırlık işlemler için, kodun anlaşılmasını kolaylaştırır.
- **Bakım Kolaylığı:** Gereksiz kod blokları olmadan, metotların işlevi daha hızlı anlaşılır ve gelecekte düzenlenmesi daha kolay olur.

Nasıl Yazılır?

Normal bir metot tanımını, expression-bodied metot şeklinde nasıl dönüştürebileceğinizi örneklerle görelim:

Geleneksel Yazım:

```
// Normal metot yazımı
```

```
static string GetGreeting()
```

```
{
```

```
    return "Merhaba, Dünya!";
```

}

Expression-Bodied Yazım:

1. static string GetGreeting() => "Merhaba, Dünya!";

Görüldüğü gibi, metodun gövdesinde sadece tek bir ifade olduğunda süslü parantezler ve return ifadesine ihtiyaç kalmadan => operatörü kullanılarak daha kısa bir tanım yapılabilir.

```
class Program
{
    static void Main()
    {
        GetGreeting();
    }
    static string GetGreeting() => "Merhaba, Dünya!";
}
```

Şekil 7 Expression Bodied Örneği

Detaylı Açıklama:

- **Ok Operatörü (=>):**
Bu operatör, metodun imzası ile metodun yaptığı işlemi birbirine bağlar. Ok operatörü kullanılarak, "burada şunu hesapla ve sonucu döndür" demek daha kolay ifade edilir.
Örneğin:

```
static int Square(int number) => number * number;
```

Bu örnekte, Square metodu verilen sayının karesini hesaplar ve sonucu döndürür.

- **Kullanım Alanları:**
Expression-bodied metotlar, özellikle şu durumlarda tercih edilir:
 - Metodun sadece tek bir hesaplama veya dönüş işlemi yapması.
 - Lambda ifadelerine benzer, okunması kolay ve kısa kod parçacıkları oluşturmak.
 - Kodun daha modern ve şık görünmesini sağlamak.
- **Sınırlamalar:**
Eğer metodun içinde birden fazla işlem veya koşul bulunuyorsa, expression-bodied metotlar yerine geleneksel metod yazımını kullanmak daha uygundur. Çünkü expression-bodied metotlar yalnızca tek bir ifade ile sınırlıdır.

Bu metodlar birden fazla satır ile yazılabilir. Bu durumda return ifadesi kullanılmaz. Size ileride bu konu daha detaylı anlatılacaktır.

TEST

Soru 1: Metotların kullanılmasının en temel avantajı nedir?

- A) Kod tekrarını artırmak
- B) Kod tekrarını azaltmak
- C) Programın hızını düşürmek
- D) Programın boyutunu artırmak

Soru 2: C# dilinde metotlar hangi yapı içerisinde tanımlanmalıdır?

- A) Döngüler içinde
- B) Sınıflar (class) içinde
- C) Değişkenler içinde
- D) Diziler içinde

Soru 3: Aşağıdakilerden hangisi metot tanımlama yapısının bileşenlerinden değildir?

- A) Erişim belirteci
- B) Metot adı
- C) Parametre listesi
- D) Döngü yapısı

Soru 4: Bir metodun herhangi bir değer döndürmediğini belirtmek için hangi anahtar kelime kullanılır?

- A) int
- B) return
- C) void
- D) static

Soru 5: Metot parametreleri tanımlanırken parametre sırası neden önemlidir?

- A) Argümanların parametrelerle eşleşmesi için
- B) Veri türü belirlemek için
- C) Bellek yönetimi için
- D) Dönüş tipi belirlemek için

Soru 6: Metotlarda "ref" anahtar kelimesi ne işe yarar?

- A) Parametreyi metot içerisinde değiştirilemez hale getirir
- B) Parametrenin değerini kopyalar
- C) Parametrenin adresini metoda iletir ve değişikliğin geri yansımaları sağlar
- D) Parametre değerini null yapar

Soru 7: Aşağıdaki anahtar kelimelerden hangisi, metodun birden fazla geriye değer döndürmesine imkan tanır?

- A) ref
- B) out
- C) in
- D) params

Soru 8: "params" anahtar kelimesi hakkında aşağıdaki ifadelerden hangisi doğrudur?

- A) Bir metotta birden fazla kez kullanılabilir
- B) Metodun başında yer alabilir
- C) Yalnızca integer değerlerle kullanılabilir
- D) Değişken sayıda parametre alan metotlar oluşturmak için kullanılır

Soru 9: Recursive (özyinelemeli) metotların temel özelliği nedir?

- A) Parametre almamaları
- B) Birden fazla değer döndürmeleri
- C) Kendi kendilerini çağırmaları
- D) Metot overload edilmesi

Soru 10: Metot overloading (aşırı yükleme) için aşağıdaki koşullardan hangisi doğrudur?

- A) Aynı isimde metotlar tanımlanamaz
- B) Metotların dönüş tipleri farklı olmalıdır
- C) Her metot sadece bir kez overload edilebilir
- D) Metotların parametre sayısı veya türü farklı olmalıdır

Soru 11: Local functions (yerel metotlar) kullanımının temel faydası nedir?

- A) Kodun daha düzenli ve okunabilir olmasını sağlar
- B) Kodun tekrar kullanılmasını engeller
- C) Kodun okunabilirliğini azaltır
- D) Performansı düşürür

Soru 12: Expression-bodied metotların temel amacı nedir?

- A) Tek satırlık metotları daha kısa ve okunabilir hale getirmek
- B) Uzun ve detaylı metotlar oluşturmak
- C) Metotları karmaşık hale getirmek
- D) Birden fazla ifade içeren metotlar oluşturmak

CEVAP ANAHTARI

1. B
2. B
3. D
4. C
5. A
6. C
7. B
8. D
9. C
10. D
11. A
12. A

Öğuzhan KARAGÜZEL