

# C# Programlama

Lambda İfadeleri ve LINQ

Emir Öztürk

# Lambda İfadesi

- $\Rightarrow$  operatörü
- (Parametreler) $\Rightarrow$ ifade
- Parametreler $\Rightarrow$ {durumlar}
- Func
- Action
- Predicate
- Fonksiyon parametre almak için

# Lambda İfadesi

## Func - Action - Predicate

- Func
  - >=0 parametre
  - Dönüş değeri
- Action
  - >=0 parametre
  - Değer döndürmez
- Predicate
  - Bir parametre (Çok parametre için tuple)
  - Bool kontroller için

```
Func<int, int, int> Topla = (x, y) => x + y;  
  
Action<int,int> ToplaGoster = (x, y) => Console.WriteLine(x + y);  
  
Predicate<int> Topla10danBuyukMu = new Predicate<int>(x=>x > 10);
```

# Lambda İfadeleri

## ifade

```
int ToplaFonksiyon(int x, int y)
{
    return x + y;
}
```



```
Func<int, int, int> Topla = (x, y) => x + y;
```


# Lambda İfadesi

```
class Program {  
    3 usages  
    public static int Uygula(int x, int y, Func<int, int, int> fonksiyon)  
    {  
        return fonksiyon(x, y);  
    }  
    public static void Main(string[] args)  
    {  
        List<Kisi> kisiListesi = new List<Kisi>();  
  
        Func<int, int, int> Topla = (x, y) => x + y;  
        Func<int, int, int> ToplaCokSatir = (x, y) =>  
        {  
            int sonuc = x + y;  
            return sonuc;  
        };  
        Func<int, int, double> UsAl = (x, y) => Math.Pow(x, y);  
  
        int x = 10;  
        int y = 5;  
        Console.WriteLine(Uygula(x, y, Topla));  
        Console.WriteLine(Uygula(x, y, ToplaCokSatir));  
        Console.WriteLine(Uygula(x, y, UsAl));  
    }  
}
```

Argument type 'System.Func<int,int,double>' is not assignable to parameter type 'System.Func<int,int,int>'

local variable `Func<int,int,double> UsAl`

# Lambda İfadesi

```
class Program {  
     3 usages  
    public static T Uygula<T>(int x, int y, Func<int, int, T> fonksiyon)  
    {  
        return fonksiyon(x, y);  
    }  
    public static void Main(string[] args)  
    {  
        List<Kisi> kisiListesi = new List<Kisi>();  
  
        Func<int, int, int> Topla = (x, y) => x + y;  
        Func<int, int, int> ToplaCokSatir = (x, y) =>  
        {  
            int sonuc = x + y;  
            return sonuc;  
        };  
        Func<int, int, double> UsAl = (x, y) => Math.Pow(x, y);  
  
        int x = 10;  
        int y = 5;  
        Console.WriteLine(Uygula(x, y, Topla));  
        Console.WriteLine(Uygula(x, y, ToplaCokSatir));  
        Console.WriteLine(Uygula(x, y, UsAl));  
    }  
}
```

# LINQ

## Koleksiyonlar

- Koleksiyonların üzerinde LINQ sorguları kullanılabilir.
- Koleksiyonlar sınıf türünden olabilir.
- Sınıf türünde olduğunda sorgunun atılacağı element lambda ifadeleri ile belirtilir.

# LINQ

## Koleksiyonlar

- Temel türden koleksiyonlarda parametresiz bazı metotlar kullanılabilir.
- Min()
- Max()
- Average()



# LINQ

## Koleksiyonlar

- Koleksiyonlardan belirli elemanları almak için metotlar bulunur.
- First()
- FirstOrDefault()
- Last()
- LastOrDefault()
- Skip()
- Take()

# LINQ

## Select

- Bir metodun koleksiyon üzerinde uygulanması için Select kullanılır.
- Metot lambda ifadesi olabilir.
- Dönüş türü IEnumerable<T>

```
static int Xor1(int x) { return x ^ 1; }

public static void Main()
{
    List<int> sayilar = new List<int>{0,0,1,1,0,1,0,1,1,0,0,1,1,0,1,1,0};
    var sonucFonk:IEnumerable<int> = sayilar.Select(Xor1);
    var sonuc:IEnumerable<int> = sayilar.Select(x:int => x ^ 1);
}
```

# LINQ

## Where

- Bir koşulun koleksiyon üzerinde kontrolü için Where kullanılır.
- Koşulu sağlayan elemanlar döndürülür.
- Dönüş türü IEnumerable<T>

```
public static void Main()
{
    List<int> sayilar = new List<int>{0,0,1,1,0,1,0,1,1,0,0,1,1,0,1,1,0};
    var sonucFonk:IEnumerable<int> = sayilar.Where(x:int =>x>0);
    var sonuc:IEnumerable<int> = sayilar.Select(x:int => x ^ 1);
}
```

# LINQ

## Any ve All

- Verilen koşulun koleksiyon içerisinde en az bir eleman tarafından sağlanma kontrolü için any
- Verilen koşulun koleksiyon içerisinde tüm elemanlar tarafından sağlandığının kontrolü için all
- Dönüş türü bool

```
List<int> sayilar = new List<int>{0,0,1,1,0,1,0,1,1,0,0,1,1,0,1,1,0};  
var sonucAny:bool = sayilar.Any(x:int =>x>0);  
var sonucAll:bool = sayilar.All(x:int =>x>0);
```

# LINQ

## Order By

- Sıralama işlemleri için
  - OrderBy
  - OrderByDescending
- Sıralama sınıf koleksiyonları olması durumlarında alana göre lambda ifadesi ile yapılabilir.
- Dönüş türü `IOrderedEnumerable<T>`

```
List<Kisi> kisiler = new List<Kisi>();  
var adaGore :IOrderedEnumerable<Kisi> = kisiler.OrderBy(x:Kisi => x.Ad);  
var soyadaGoreAzalan :IOrderedEnumerable<Kisi> = kisiler.OrderByDescending(x:Kisi => x.Soyad);
```

# LINQ

## Lambda ifadelerini birleştirme

- Lambda ifadeleri art arda çağırılabilir.
- Önce seçme, sonra sıralama, daha sonra belirli sayıda alma gibi bir işlem gerçekleştirilebilir.

```
List<Kisi> kisiler = new List<Kisi>();  
var ilk5Listesi:List<string> = kisiler.Where(x:Kisi => x.Ad.StartsWith("A")) // IEnumerable<Kisi>  
    .OrderBy(x:Kisi => x.Ad) // IOrderedEnumerable<Kisi>  
    .Take(5) // IEnumerable<Kisi>  
    .Select(x:Kisi => x.Ad + " " + x.Soyad) // IEnumerable<string>  
    .ToList();
```

# LINQ

- Lambda ifadeleri yerine LINQ syntax'ı da kullanılabilir.

```
List<Kisi> kisiler = new List<Kisi>();  
var ilk5Listesi:List<string> = kisiler.Where(x:Kisi => x.Ad.StartsWith("A")) // IEnumerable<Kisi>  
    .OrderBy(x:Kisi => x.Ad) // IOrderedEnumerable<Kisi>  
    .Take(5) // IEnumerable<Kisi>  
    .Select(x:Kisi => x.Ad + " " + x.Soyad) // IEnumerable<string>  
    .ToList();
```

```
List<Kisi> kisiler = new List<Kisi>();  
var sonuc:List<string> = (from x:Kisi in kisiler  
    where x.Ad.StartsWith("A")  
    orderby x.Ad  
    select x.Ad + " " + x.Soyad  
).Take(5).ToList();
```