</> C# · Orta



C# — Orta Seviye

Koleksiyonlar, hata yönetimi, LINQ ve ileri OOP ile C#’ta bir üst seviye.

SowaCodeHK — Ders Notları
sowacodehk.com.tr · Hazar Kanat

Bu notta ne öğreneceksin? C#'ın bir üst seviyesi: koleksiyonlar (List, Dictionary), hata yönetimi (try-catch), LINQ ile veri sorgulama, interface ve ileri OOP kavramları (abstract, polymorphism). "C#'a Giriş" notunu bitirdiysen buraya hazırsın.

Koleksiyonlar

Birden çok veriyi bir arada tutmak için diziden daha esnek yapılar kullanılır. En yaygını **List**:

```
List<string> isimler = new List<string>();
isimler.Add("Hazar");
isimler.Add("Ayşe");

Console.WriteLine(isimler.Count);   // 2
Console.WriteLine(isimler[0]);      // Hazar

foreach (string isim in isimler) {
    Console.WriteLine(isim);
}
```

Dictionary ise anahtar–değer çiftleri tutar:

```
Dictionary<string, int> yaslar = new Dictionary<string, int>();
yaslar["Hazar"] = 25;
yaslar["Ayşe"] = 30;

Console.WriteLine(yaslar["Hazar"]);   // 25
```

İpucu: Sabit boyutlu `array` yerine çoğu zaman `List` kullan; eleman ekleyip çıkarmak çok daha kolaydır.

Hata Yönetimi (try - catch)

Program çökmeden hataları yakalamak için kullanılır. Örneğin metni sayıya çevirirken hata olabilir:

```
try {
    int sayi = int.Parse("abc");   // hata fırlatır
    Console.WriteLine(sayi);
}
```

```
catch (Exception ex) {
    Console.WriteLine("Hata oluştu: " + ex.Message);
}
finally {
    Console.WriteLine("Bu blok her durumda çalışır.");
}
```

- `try` — Riskli kod buraya.
- `catch` — Hata olursa burası çalışır, program çökmez.
- `finally` — Hata olsun olmasın her zaman çalışır (isteğe bağlı).

LINQ ile Veri Sorgulama

LINQ, koleksiyonları SQL benzeri sorgularla filtreleyip dönüştürmeni sağlar. C#'ın en güçlü özelliklerinden biridir:

```
List<int> sayilar = new List<int> { 5, 12, 8, 130, 44, 3 };

// 10'dan büyük olanları seç
var buyukler = sayilar.Where(s => s > 10).ToList();
// Sonuç: 12, 130, 44

// Küçükten büyüğe sırala
var sirali = sayilar.OrderBy(s => s).ToList();

// Toplamları
int toplam = sayilar.Sum();
int enBuyuk = sayilar.Max();
```

Nesne listelerinde de çalışır:

```
var gencler = kisiler
    .Where(k => k.Yas < 30)
    .OrderBy(k => k.Ad)
    .ToList();
```

> **İpucu:** `s => s > 10` ifadesine "lambda" denir. "Her s için, s 10'dan büyük mü?" demektir. LINQ'in kalbi budur.

Interface (Arayüz)

Interface, bir sınıfın *ne yapması gerektiğini* belirleyen sözleşmedir; *nasıl yapacağını* sınıf kendi doldurur. Ortak davranışları standartlaştırır:

```
interface ISekil {
    double Alan();   // sadece imza — gövde yok
}

class Kare : ISekil {
    public double Kenar;
    public double Alan() {
        return Kenar * Kenar;   // kendi hesabı
    }
}

class Daire : ISekil {
    public double Yaricap;
    public double Alan() {
        return 3.14 * Yaricap * Yaricap;
    }
}
```

İleri OOP: abstract ve polymorphism

abstract sınıf, doğrudan nesne üretilemeyen, miras için tasarlanmış temel sınıftır. **Polymorphism (çok biçimlilik)** ise aynı metodun farklı sınıflarda farklı davranmasıdır:

```
abstract class Hayvan {
    public abstract void SesCikar();   // her hayvan farklı yapacak
}

class Kopek : Hayvan {
    public override void SesCikar() {
        Console.WriteLine("Hav!");
    }
}
class Kedi : Hayvan {
    public override void SesCikar() {
        Console.WriteLine("Miyav!");
    }
}

// Aynı çağrı, farklı davranış (polymorphism)
List<Hayvan> hayvanlar = new List<Hayvan> { new Kopek(), new Kedi() };
foreach (var h in hayvanlar) {
```

```
    h.SesCikar();   // Hav! ... Miyav!
}
```

Özet

- `List` ve `Dictionary` esnek veri saklama sağlar.
- `try-catch` ile program çökmeden hatalar yakalanır.
- LINQ (`Where, OrderBy, Select, Sum`) ile veriler güçlü şekilde sorgulanır.
- Interface ortak davranış sözleşmesi kurar; abstract ve polymorphism ileri OOP'nin temelidir.
- Sıradaki adım: **ASP.NET Core** ile bu bilgiyi web uygulamasına dökmek.