Bu dokümanda, sağlam ve güvenilir C# uygulamaları geliştirmenin temel taşlarından biri olan hata yönetimi (exception handling) mekanizması ele alınmaktadır. Bir programın kalitesi, yalnızca doğru çalıştığı senaryolarla değil, aynı zamanda beklenmedik durumlar ve hatalar karşısındaki tepkisiyle ölçülür. Bu çalışma, C#'ta istisnaların (Exception) ne olduğunu, program akışını nasıl kesintiye uğrattığını ve try-catch-finally blokları kullanılarak bu istisnaların nasıl kontrollü bir şekilde yönetileceğini detaylandırmaktadır. throw anahtar kelimesi ile kendi özel hata durumlarımızı nasıl oluşturacağımızı da kapsayan bu rehber, programların beklenmedik durumlarda çökmesini önlemek ve daha kararlı yazılımlar üretmek isteyen her C# geliştiricisi için temel bir kaynaktır.

HATA YÖNETİMİ VE İSTİSNALAR (EXCEPTIONS)

try-catch-finally

Oğuzhan Karagüzel

İçindekiler

GİRİŞ

Bu dokümanda, C# programlama dilinde hata yönetimi (exception handling) mekanizmasını detaylı olarak inceliyoruz. Bir uygulamanın kararlılığı ve güvenilirliği, yalnızca beklenen durumlarda doğru çalışmasıyla değil, aynı zamanda beklenmedik durumlar ve hatalar karşısında nasıl davrandığıyla da ölçülür. Bir dosyanın bulunamaması, ağ bağlantısının kopması veya kullanıcının geçersiz veri girişi yapması gibi senaryolar, programın normal akışını bozan istisnai durumlardır.

C#'ta bu istisnai durumlar, **Exception** adı verilen nesneler aracılığıyla yönetilir. Eğer bu nesneler ele alınmazsa (unhandled exception), programın çalışması sonlanır. Bu doküman, C#'ın yapısal hata yönetimi mekanizması olan try-catch-finally bloğunu kullanarak bu istisnaların nasıl yönetileceğini, hataların nasıl yakalanacağını ve program akışının nasıl kontrol altında tutulacağını açıklamak amacıyla hazırlanmıştır.

EXCEPTION (İSTİSNA) KAVRAMI

C# dilinde **Exception (İstisna)**, programın çalışması sırasında meydana gelen ve normal kod akışını kesintiye uğratan bir hata veya anormal durumu temsil eden bir nesnedir. Bu nesneler, .NET kütüphanesinde önceden tanımlanmış veya kullanıcı tarafından oluşturulmuş sınıflardan türetilir. Nesne tabanlı programlamada bunlara daha detaylı değineceğiz. Ancak try catch yapısını şimdi öğrenmeniz gerekiyor. Bundan dolayı exception kullanımını temel seviyede tutacağız.

Her istisna tipi, belirli bir hata senaryosunu ifade eder:

- **FormatException:** Bir string (metin), hedeflenen formata (örneğin sayıya) dönüştürülemediğinde oluşur.
- **DivideByZeroException:** Bir sayının sıfıra bölünmeye çalışılması durumunda oluşur.
- **FileNotFoundException:** Erişilmeye çalışılan bir dosya belirtilen yolda bulunamadığında oluşur.
- **ArgumentNullException:** Bir metoda null olmaması gereken bir argüman null olarak gönderildiğinde oluşur.

Bu istisnalar yönetilmediği takdirde, .NET çalışma zamanı (CLR) programı sonlandırır. try-catch mekanizmasının temel amacı, bu istisnaları programcı tarafından ele alarak kontrollü bir akış sağlamaktır.

try BLOĞU: İSTİSNA OLASILIĞI TAŞIYAN KOD KAPSÜLÜ

try bloğu, bir veya daha fazla istisna fırlatma potansiyeli olan kodları çevrelemek için kullanılır. Bu blok, derleyiciye içindeki kodların denetim altında yürütülmesi gerektiğini bildirir.

- try bloğundaki tüm kodlar hatasız bir şekilde tamamlanırsa, catch blokları atlanır ve program akışı normal bir şekilde devam eder.

- Eğer try bloğunun herhangi bir satırında bir istisna oluşursa, o satırdan sonraki kodların yürütülmesi derhal durdurulur ve kontrol, uygun olan ilk catch bloğuna geçer.

```
try
{
    // İstisna oluşturma potansiyeli olan kodlar bu kapsül içine yazılır.
    // Örneğin, veritabanı bağlantısı, dosya okuma işlemi veya riskli matematiksel hesaplamalar.
}
```

catch BLOĞU: İSTİSNALARI YAKALAMA VE ELE ALMA

catch bloğu, try bloğunda meydana gelen bir istisnayı "yakalamak" ve bu duruma yanıt vermek için kullanılır. Hangi türde bir istisnanın yakalanacağı parantez içinde belirtilir.

En genel yakalama bloğu, tüm istisna tiplerinin türediği temel sınıf olan Exception sınıfını kullanır: catch (Exception ex)

Buradaki ex değişkeni, yakalanan istisna nesnesinin bir referansıdır ve hata hakkında detaylı bilgiler içerir:

- **ex.Message (string):** İstisnanın ne olduğunu açıklayan kısa, okunabilir bir metin döndürür. (Örn: "Input string was not in a correct format.")
- **ex.StackTrace (string):** Hatanın hangi metotta, hangi satırda ve hangi çağrı zinciri sonucunda oluştuğunu gösteren teknik bir rapor sunar. Geliştiriciler için hatanın kaynağını bulmada kritik öneme sahiptir.

Örnek: Kullanıcı Girdisini Sayıya Çevirme

```
Console.WriteLine("Lütfen yaşınızı giriniz:");
string girdi = Console.ReadLine();

try
{
    Console.WriteLine("-> try bloğu başladı.");
    int yas = int.Parse(girdi); // Bu satırda istisna oluşabilir.
    Console.WriteLine($"Girdiğiniz yaş: {yas}. Bu satıra hata olmazsa ulaşılır.");
}
catch (Exception ex)
{
```

```
    Console.WriteLine("-> catch bloğu çalıştı, çünkü bir istisna
yakalandı.");
    Console.WriteLine($"Hata Mesajı: {ex.Message}");
}
Console.WriteLine("-> Program çalışmaya devam ediyor.");
```

ÇALIŞMA SENARYOLARI:

1. **Hatasız Senaryo (Girdi: "30")**

Lütfen yaşınızı giriniz:

30

-> try bloğu başladı.

Girdiğiniz yaş: 30. Bu satıra hata olmazsa ulaşılır.

-> Program çalışmaya devam ediyor.

catch bloğu hiç çalışmadı, program normal akışına devam etti.

2. **Hatalı Senaryo (Girdi: "otuz")**

Lütfen yaşınızı giriniz:

otuz

-> try bloğu başladı.

-> catch bloğu çalıştı, çünkü bir istisna yakalandı.

Hata Mesajı: The input string 'otuz' was not in a correct format.

-> Program çalışmaya devam ediyor.

int.Parse satırında bir FormatException oluştuğu anda try bloğunun geri kalanı atlandı ve kontrol doğrudan catch bloğuna devredildi. Program çökmedi.

finally BLOĞUNUN AMACI: KAYNAK TEMİZLİĞİ

finally bloğu, bir istisna oluşup oluşmadığına bakılmaksızın, try-catch yapısından çıkılmadan hemen önce **her durumda çalışması garanti edilen** kod bloğudur.

Temel amacı, try bloğunda açılmış olan kaynakların (dosya akışları, veritabanı bağlantıları, ağ soketleri vb.) güvenli bir şekilde kapatılmasını ve serbest bırakılmasını sağlamaktır. Buna "kaynak temizliği" (resource cleanup) denir.

- Hata oluşmazsa, try bittikten sonra finally çalışır.
- Hata oluşur ve catch ile yakalanırsa, catch bittikten sonra finally çalışır.

```
try
{
    Console.WriteLine("try: Kaynaklar açılıyor.");
    // Kod...
}
catch (Exception ex)
{
    Console.WriteLine("catch: İstisna ele alınıyor.");
}
finally
{
    // Bu blok hata olsun veya olmasın her zaman çalışır.
    Console.WriteLine("finally: Kaynaklar temizleniyor ve kapatılıyor.");
}
```

finally bloğunun varlığı, uygulamanın kaynak sızıntısı yapmasını engelleyerek daha stabil çalışmasını sağlar.

Şu anda kaynak kullanımı size biraz yabancı geliyor olabilir. Bunu ileride daha iyi anlayacaksınız.

İSTİSNA FIRLATMA: throw ANAHTAR KELİMESİ

Programcılar, yalnızca .NET tarafından fırlatılan istisnaları yakalamakla kalmaz, aynı zamanda kendi iş mantıklarına uymayan durumlar için bilinçli olarak istisna fırlatabilirler. Bu işlem, throw anahtar kelimesi ile yapılır.

Bu, metodun kendi sözleşmesini ihlal eden durumları, çağıran koda bir hata olarak bildirmesini sağlar.

```
public void BankaHesabindanParaCek(decimal miktar)
{
    decimal mevcutBakiye = 100m; // Örnek bakiye
    if (miktar > mevcutBakiye)
    {
        // İş mantığına aykırı bir durum: yetersiz bakiye.
        // Bu durumu bir istisna olarak fırlatıyoruz.
        throw new InvalidOperationException("Yetersiz bakiye. İşlem
gerçekleştirilemedi.");
```

```
    }
    // İşlem devam eder...
}

// ---- Kullanım ----
try
{
    BankaHesabindanParaCek(200m); // İstisna fırlatacak.
}
catch (InvalidOperationException ex)
{
    Console.WriteLine($"İşlem Hatası: {ex.Message}");
}
// Çıktı: İşlem Hatası: Yetersiz bakiye. İşlem gerçekleştirilemedi.
```

BİRDEN FAZLA catch BLOĞU KULLANIMI

Bir try bloğunda birden farklı türde istisna oluşabilir. Her bir istisna türü için farklı bir eylem gerçekleştirmek isteniyorsa, birden fazla catch bloğu tanımlanabilir.

Bu durumda C#, oluşan istisna türüyle eşleşen ilk catch bloğunu çalıştırır. catch blokları, **en özel istisnadan en genel istisnaya doğru** sıralanmalıdır. Çünkü C# yukarıdan aşağıya doğru ilk eşleşmeyi arar. catch (Exception ex) bloğu her zaman en sona konulmalıdır.

```
try
{
    Console.WriteLine("Bir sayı giriniz:");
    string girdi = Console.ReadLine();
    int sayi = int.Parse(girdi);
}
catch (FormatException)
{
    // Yalnızca format hatası oluştuğunda çalışır.
    Console.WriteLine("Geçersiz format. Lütfen yalnızca rakam giriniz.");
}
```

```
catch (OverflowException)
{
    // Yalnızca sayı int'in sınırları dışındaysa çalışır.
    Console.WriteLine("Girilen sayı çok büyük veya çok küçük.");
}
catch (Exception ex)
{
    // Yukarıdakilerin dışındaki tüm diğer istisnalar için çalışır.
    Console.WriteLine($"Beklenmedik bir hata oluştu: {ex.Message}");
}
```

ÖZET

1. **İstisna (Exception)**, programın normal akışını bozan, nesne olarak temsil edilen bir hata durumudur. Yönetilmezse program sonlanır.
2. **try-catch** mekanizması, istisna potansiyeli taşıyan kodları denetler (try), bir istisna oluştuğunda bunu yakalar (catch) ve programın çökmesini engelleyerek kontrollü bir şekilde hatayı ele almamızı sağlar.
3. **finally** bloğu, bir hata meydana gelse de gelmese de, her koşulda çalıştırılması garanti edilen kodları barındırır ve genellikle kaynak temizliği için kullanılır.