EXCEPTION

`Exception`, bir programın çalışma zamanında ortaya çıkan ve normal program akışını bozan hataları temsil eden bir türdür. Programların çalıştığı sırada bir hata meydana geldiğinde, bu hata bilgilerini içeren bir `Exception` nesnesi oluşturulur ve bu nesne program içinde bir hata durumunu temsil eder.

`Exception` sınıfı, .NET platformunda tüm istisna türlerinin temel sınıfıdır ve diğer istisna türleri bu sınıftan türetilir. `Exception` sınıfı, birçok özellik ve metot içerir, bu nedenle farklı türdeki hataları temsil etmek ve işlemek için geniş bir kullanım yelpazesine sahiptir.

`Exception` sınıfının bazı önemli özellikleri şunlardır:

1. Message: Hata ile ilgili açıklamayı içeren bir metin dizesidir. Genellikle hatanın sebebini açıklar.

2. StackTrace: Hatanın meydana geldiği yerin izini içeren bir dizedir. Bu, hatanın hangi metotlardan geçtiğini gösterir. `ex.StackTrace` ifadesi, hatanın oluştuğu metotlardan geçişlerin sırasını içeren bir dize döndürür. Bu, hatanın neden kaynaklandığını anlamak ve hatayı gidermek için kullanışlı bir bilgidir.

3. InnerException: Bir istisna diğer bir istisna tarafından tetiklenirse, bu özellik iç içe geçmiş istisnaların bilgisini içerir.

İstisna mekanizması, programların daha sağlam ve hata toleranslı olmasına yardımcı olur. Hataların kontrol edilebilir ve yönetilebilir olması, uygulamaların daha güvenilir olmasına katkıda bulunur. Programcılar, `try`, `catch`, ve `finally` bloklarını kullanarak istisnaları yakalayabilir, işleyebilir ve uygun bir şekilde tepki verebilir.

THROW

throw anahtar kelimesi, bir programın çalışma sırasında bir hata durumu ortaya çıktığında, bu hatayı programın dışındaki bir koda bildirmek ve işlemek için kullanılır. **throw** anahtar kelimesi, bir istisna (exception) nesnesini başlatmak veya mevcut bir istisna nesnesini tekrar fırlatmak için kullanılır. Bu, programın kontrol akışını değiştirmeye ve hatanın uygun şekilde ele alınmasını sağlamaya yardımcı olur.

Exception türlerine girmeden önce şu soruyu soralım; neden hataları barındıran exception sınıfını kullanarak hata ayıklamak yerine bu sınıfın içerisinde bulunan spesifik (özelleşmiş) hataları yakalamalı veya fırlatmalıyız?

.NET'te genel bir kural olarak, spesifik istisna türlerini kullanmak, kodunuzun daha okunabilir, bakımı daha kolay ve hataları daha etkili bir şekilde yönetebilmeniz açısından genellikle daha iyi bir uygulama tasarımına yol açar. İşte bu tercihin bazı nedenleri:

1. Daha Spesifik Bilgi: Spesifik istisna türleri, hatanın nedenini daha iyi tanımlar. Örneğin, bir `FileNotFoundException` dosya bulunamadığında ortaya çıkabilir. Bu, hatanın kökenini daha iyi anlamanızı ve hata durumlarına daha iyi tepki vermenizi sağlar.

2. Daha İyi Hata Yakalama ve İşleme: Spesifik istisna türlerini kullanmak, sadece belirli hata durumlarına odaklanmanıza ve bu durumlar için özel bir işlem yapmanıza olanak tanır. Bu, hatayı daha etkili bir şekilde ele almanızı sağlar.

3. Kodun Daha Okunabilir Olması: Spesifik istisna türleri kullanmak, kodunuzun okunabilirliğini artırır. Kodunuzun okunması ve anlaşılması daha kolaydır, çünkü hangi hatalara odaklandığınız açıkça belirtilir.

4. Hata Türlerini Daha İyi Belgeleme: Spesifik istisna türlerini kullanmak, kodunuzun kullanıcıları veya diğer geliştiriciler için hata durumları hakkında daha iyi belgelendirilmiş bilgiler sunmanıza yardımcı olur. Bu, hata durumlarını anlamak ve ele almak için daha kolay bir referans sağlar.

5. Yapılandırılmış Hata İşleme: Spesifik istisna türleri, hata durumlarına yapılandırılmış bir şekilde yaklaşmanıza ve işlemenize olanak tanır. Örneğin, bir dosya bulunamadığında özel bir işlem gerçekleştirmek istiyorsanız, `FileNotFoundException`'ı yakalayabilir ve buna özel bir tepki verebilirsiniz.

Ancak, her durumda genel bir `Exception` sınıfını kullanmak uygun olabilir, özellikle beklenmeyen hata durumlarını ele almak için. Ancak, genel bir `Exception` kullanılırken, bu durumu mümkün olduğunca spesifik hale getirmek için içerideki `InnerException` özelliğini kullanmak önemlidir.

Exception Türleri

AccessViolationException: **AccessViolationException**, bir uygulama tarafından bellek erişimi ihlali durumunda fırlatılan bir özel bir **SystemException** türüdür. Bu tür bir exception, genellikle geçersiz bir bellek adresine erişim girişiminde bulunduğunda ortaya çıkar. Bu örnek sadece anlamak için, gerçek projelerde böyle bir şey yapmamalısınız. Unutulmamalıdır ki bu tür hataların bilinçli olarak oluşturulması ve kullanılması önerilmez. Gerçek uygulamalarda hafıza erişimi güvenli bir şekilde yönetilmelidir.

```
try
{
    // Bellek erişimi ihlali oluşturacak bir kod örneği
    // Bu örnek sadece anlamak için, gerçek projelerde böyle bir şey yapmamalısınız.
    int[] arr = null;
    Console.WriteLine(arr[0]);
}
catch (AccessViolationException ex)
{
    MessageBox.Show("AccessViolationException yakalandı:\n" + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch (Exception ex)
{
    MessageBox.Show("Beklenmeyen bir hata oluştu:\n" + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

Yukarıdaki hata null (herhangi bir değere sahip olmadığı için boş kabul edilir) tanımlanan bir dizinin birinci elemanına ulaşılmaya çalışılmasından kaynaklanmaktadır. Bu hata ayıklama yöntemi compiler tarafından direkt bulunduğu için programın çalışmasını engeller ve durdurur. Bu nedenle aşağıda 'throw' ile hata fırlatma örneği yazdım. Bu şekilde herhangi bir durumda belleğe veya verilere ulaşılmaya çalışılması gibi riskli durumlarda ve güvenlik açısından ihtiyaç duyulan yerlerde aşağıdaki şekilde hata fırlatılabilir.

```
1 reference
private void button1_Click(object sender, EventArgs e)
{

        try
        {
            // Bellek erişimi ihlali oluşturacak bir kod örneği
            // Bu örnek sadece anlamak için, gerçek projelerde böyle bir şey yapmamalısınız.
            throw new AccessViolationException("Bu özel bir AccessViolationException hatası!");
        }
        catch (AccessViolationException ex)
        {
            MessageBox.Show("AccessViolationException yakalandı:\n" + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Beklenmeyen bir hata oluştu:\n" + ex.Message, "Hata", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
```

ArgumentException & ArgumentNullException

`ArgumentException` ve `ArgumentNullException`, .NET'te kullanılan spesifik istisna türlerindendir. Her ikisi de metodlara geçirilen argümanların geçerliliğini kontrol etmek için kullanılır. İşte her iki istisna türü hakkında daha fazla bilgi ve örnekler:

ArgumentException:

- `ArgumentException`, bir metodun parametrelerinden biri veya birkaçı geçerli olmadığında fırlatılan genel bir istisna türüdür.

- Bu hatanın temel nedeni, metodun beklediği argüman değerlerinin belirli bir kriteri karşılamamasıdır.

Aşağıdaki örnekte negatif argüman girilmesini istemediğimiz bir durumda değerin negatif olup olmadığını kontrol ederek hata fırlatma örneği verilmektedir. Bu şekilde aldığımız argümanlara istediğimiz kriterlere göre ArgumentException fırlatabiliriz.

```
public void SetAge(int age)
{
    if (age < 0)
    {
        throw new ArgumentException("Yaş negatif olamaz.", nameof(age));
    }
}
```

ArgumentNullException:

- `ArgumentNullException`, bir metodun bir veya birden fazla referans tipindeki argümanının `null` (boş) olması durumunda fırlatılan istisna türüdür.

- Bu hatanın temel nedeni, metodun beklediği referans tipindeki argümanlardan birinin `null` olması durumunda gerçekleşir.

```
public void SetName(string name)
{
    if (name == null)
    {
        throw new ArgumentNullException(nameof(name), "İsim null olamaz.");
    }
}
```

Yukarıdaki örnekte, `SetName` adlı metod bir isim değeri alır ve bu değerin `null` olup olmadığını kontrol eder. Eğer isim `null` ise, `ArgumentNullException` fırlatılır.

Dosyalarla ilgili de dosyanın bulunamaması, görüntü kalitesinin kötü olması, bellek yetersiz olduğunda OutOfMemoryException , PlatformNotSupportedException, belirli bir işlemin istenilen sürede bitirilemediğinde TimeoutException gibi spesifik hata türleri ihtiyaca göre kullanılmalıdır.