**Hata Yönetimi**

* Hata yönetimi try - catch - finally blokları ile yapılır.

|  |
| --- |
| try{  //Hataya sebep olabilecek kodlar buraya yazılır.  }  catch{  //Hata oluşursa catch blogu çalışır.  }  finally{  //Hata olsa da olmasa da her şartta çalışması gereken kodlar  //için finally bloğu hazırlayabilirsiniz.  } |

* try - catch - finally blokları arasında kalan alana kod yazılmaz.
* İç içe try - catch blokları oluşturabilirsiniz.
* try - catch - finally blokları sadece metot içerisinde kullanılabilir.
* Aşağıdaki örnekte hataya sebep olan satır metot içinde olmasına rağmen try bloğu bu hatayı yakalayıp catch bloğundaki kodların çalışmasını sağlayacaktır.

|  |
| --- |
| static void Main(){  try{  Islem();  }  catch{  Console.WriteLine(“Hata meydana geldi.”); //Hata meydana geldi.  }  }  //Sıfıra bölme yaparak bir hata oluşmasını sağladık  static void Islem(){  int i = 10;  int j = 0;  Console.WriteLine(i/j);  } |

* Try bloğu içerisindeyken bir hata meydana geldiğinde hatayla ilgili bilgiler içeren bir nesne otomatikmen hazırlanıp fırlatılır. Bu hata nesnesi catch bloğunda yakalanıp hatanın türü ve sebebi öğrenilebilir.
* catch parametresiz olarak kullanıldığında hatanın sebebi ne olursa olsun arasına yazılan kodları çalıştırır. Ancak biz hatanın sebebine göre farklı kodların çalışmasını isteyebiliriz. Bu durumda birden fazla catch bloğu oluşturup farklı tipte parametreleri olmasını sağlamalıyız.

|  |
| --- |
| try{  //DivideByZeroException nesnesi hazırlanıp fırlatılacak.  Console.WriteLine(2/0);  }  catch(DivideByZeroException hata){  //DivideByZeroException nesnesi fırlatılmışsa bu blok çalışır.    //Message özelliği ile sistem tarafından oluşturulan hata  //mesajı elde ediliyor.  Console.WriteLine(“Sıfıra bölme hatası.” + hata.Message);  }  catch(FormatException hata){  //FormatException : Tip dönüşüm hatası gerçekleşmişse.  //FormatException nesnesi fırlatılmışsa bu blok çalışır.  }  catch(OverflowException hata){  //OverflowException : Değişkene alabileceğinden daha büyük bir  //değer verilmişse, yani sınır aşılmışsa.  }  catch(InvalidCastException){  //Unboxing işlemi başarısız olduğunda.  }  //Yukarıdaki hata sınıflarından daha fazlası framework’de //tanımlıdır. |

* Tüm hata sınıfları Exception sınıfının mirasçısıdır. Fırlatılacak hata nesnesinin türünü önceden kestiremiyorsanız catch blogunun parametresinin Exception tipinde olmasını sağlayabilirsiniz.
* throw anahtar kelimesi ile hata nesnesi fırlatabilirsiniz.

|  |
| --- |
| try{  throw new OverflowException(“bilgi mesajı.”);  }  catch(OverflowException hata){  Console.WriteLine(hata.Message); //bilgi mesajı.  } |

* Exception sınıfından yararlanarak kendi hata sınıflarınızı oluşturabilirsiniz. Detaylı bilgi için referans kaynaklardan yardım alın.