Academy .NET

Gestione Eccezioni

Sommario

- Cosa sono le eccezioni?
- Gestione eccezioni
- La classe **System.Exception**
- Tipi di eccezioni e loro gerarchia
- Generare eccezioni
- Best Practices

Cosa sono le Eccezioni?

Le eccezioni in .NET Framework sono l'implementazione classica del modello di eccezione OOP.

Forniscono un potente meccanismo per la gestione centralizzata di errori ed eventi insoliti.

Approccio sostitutivo orientato alla procedura, in cui ogni funzione restituisce un codice di errore.

Semplifica la costruzione e la manutenzione del codice.

Consentono l'elaborazione delle problematiche a più livelli.

Gestione eccezioni

In C # le eccezioni possono essere gestite dalla costruzione **try-catch-finally**.

```
try
{
    // Do some work that can raise an exception
}
catch (SomeException)
{
    // Handle the caught exception
}
```

I blocchi **catch** possono essere utilizzati più volte per elaborare diversi tipi di eccezione.

Gestione eccezioni - Esempio

```
static void Main()
{
    string s = Console.ReadLine();
    try
    {
        Int32.Parse(s);
        Console.WriteLine(
           "You entered valid Int32 number {0}.", s);
    }
    catch (FormatException)
    {
        Console.WriteLine("Invalid integer number!");
    }
    catch (OverflowException)
    {
        Console.WriteLine(
           "The number is too big to fit in Int32!");
    }
}
```

La classe System.Exception

Le eccezioni in .NET sono **oggetti.**

La classe **System.Exception** è la base per tutte le eccezioni nel CLR:

Contiene informazioni sulla causa dell'errore/situazione insolita

Message: descrizione testuale dell'eccezione

StackTrace: l'istantanea dello stack al momento della generazione dell'eccezione

InnerException: proprietà di eccezione precedente che ha causato la corrente eccezione. (se presente)

Proprietà Eccezioni - Esempio

```
class ExceptionsExample
{
  public static void CauseFormatException()
  {
    string s = "an invalid number";
    Int32.Parse(s);
  }
  static void Main()
  {
    try
    {
      CauseFormatException();
    }
    catch (FormatException fe)
    {
      Console.Error.WriteLine("Exception: {0}\n{1}",
        fe.Message, fe.StackTrace);
    }
  }
}
```

Proprietà Eccezioni

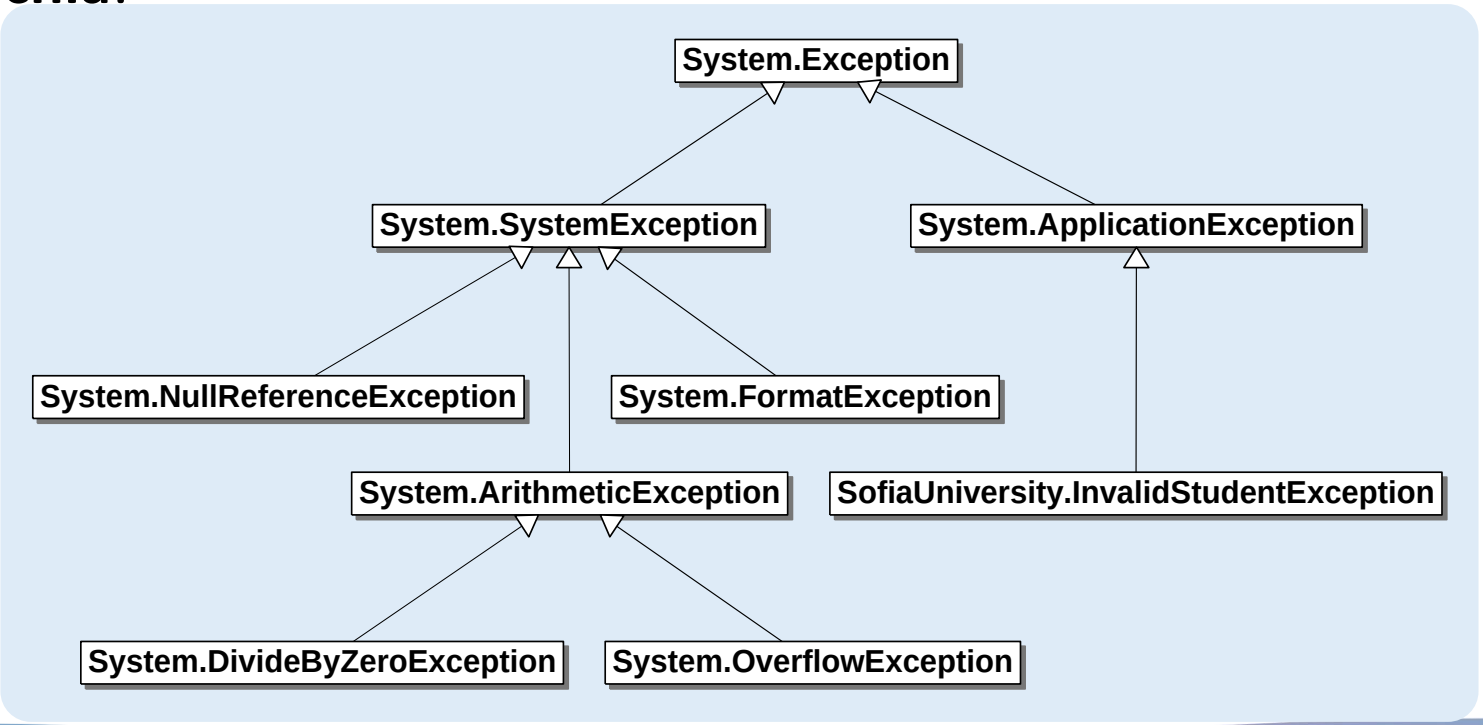
La proprietà **Message** fornisce una breve descrizione del problema.

La proprietà **StackTrace** è estremamente utile per identificare il motivo che ha causato l'eccezione.

```
Exception caught: Input string was not in a correct format.
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInf
o info)
   at System.Int32.Parse(String s)
   at ExceptionsTest.CauseFormatException() in c:\consoleapplication1\except
ionstest.cs:line 8
   at ExceptionsTest.Main(String[] args) in c:\consoleapplication1\exception
stest.cs:line 15
```

Gerarchia Eccezioni

Le eccezioni in .NET Framework sono organizzate in una **gerarchia**:



Tipi di Eccezioni

Le eccezioni .NET ereditano da **System.Exception**

Le eccezioni di sistema ereditano da **System.SystemException**, ad es.

System.ArgumentException
System.NullReferenceException
System.OutOfMemoryException
System.StackOverflowException

Le eccezioni definite dall'utente vanno ereditare da **System.Exception()**

Gestione eccezioni

Quando si cattura l'eccezione di una particolare classe, vengono catturate anche le **eccezioni ereditate**

- Esempio:

```
try
{
  // Do some works that can cause an exception
}
catch (System.ArithmeticException)
{
    // Handle the caught arithmetic exception
}
```

Gestisce **ArithmeticException** e eccezioni ereditate **DivideByZeroException** e **OverflowException**

Eccezioni – Errori da evitare

```
static void Main()
{
   string s = Console.ReadLine();
   try
   {
      Int32.Parse(s);
   }
   catch (Exception)
   {
      Console.WriteLine("Can not parse the number!");
   }
   catch (FormatException)
   {
      Console.WriteLine("Invalid integer number!");
   }
   catch (OverflowException)
   {
      Console.WriteLine(
       "The number is too big to fit in Int32!");
   }
}
```

gestire l'eccezione più generica per ultima

codice non raggiungibile

codice non raggiungibile

Gestione eccezioni

- Tutte le eccezioni generate dal codice gestito da .NET ereditano l'eccezione **System.Exception**
- Il codice non gestito può generare altre eccezioni

- Per gestire tutte le eccezioni (anche non gestite) utilizzare la costruzione:

```
try
{
    // Do some works that can raise any exception
}
catch
{
    // Handle the caught exception
}
```

Generare Eccezioni

Le eccezioni in C# vengono generate dalla parola chiave **throw**

- Utilizzata per notificare il codice chiamante in caso di errore o situazione insolita

Quando viene generata un'eccezione:

- L'esecuzione del programma si **interrompe**
- L'eccezione è gestita sullo stack finché non viene raggiunto un blocco catch adatto

Le eccezioni non gestite visualizzano un messaggio di errore.

Utilizzo parola chiave throw

- Generare un'eccezione con un messaggio di errore:

```
throw new ArgumentException("Invalid amount!");
```

Gestione Eccezioni - Esempio

```
public static double Sqrt(double value)
{
   if (value < 0)
       throw new System.ArgumentOutOfRangeException(
          "Sqrt for negative numbers is undefined!");
   return Math.Sqrt(value);
}
static void Main()
{
   try
   {
       Sqrt(-1);
   }
   catch (ArgumentOutOfRangeException ex)
   {
       Console.Error.WriteLine("Error: " + ex.Message);
       throw;
   }
}
```

Scelta del tipo di Eccezione

Quando un parametro non valido viene passato a un metodo:

ArgumentException, ArgumentNullException, ArgumentOutOfRangeException

Quando l'operazione richiesta non è supportata:

NotSupportedException

Quando un metodo non è ancora implementato:

NotImplementedException

Se non è disponibile una classe di eccezione standard adatta

Crea eccezione personalizzata (eredita Exception)

try - finally

La dichiarazione:

```
try
{
    // Do some work that can cause an exception
}
finally
{
    // This block will always execute
}
```

Assicura l'esecuzione di un blocco finally **in tutti i casi**:

Quando viene sollevata o meno un'eccezione nel blocco **try**

Utilizzato per l'esecuzione di cleaning-up code, ad es. liberare risorse.

try – finally - Esempio

```
static void TestTryFinally()
{
  Console.WriteLine("Code executed before try-finally.");
  try
  {
    string str = Console.ReadLine();
    Int32.Parse(str);
    Console.WriteLine("Parsing was successful.");
  }
  catch (FormatException)
  {
    Console.WriteLine("Parsing failed!");
  }
  finally
  {
    Console.WriteLine(
      "This cleanup code is always executed.");
  }
  Console.WriteLine(
    "This code is after the try-finally block.");
}
```

Eccezione Personalizzata : Esempio

```
using System;

public class EmployeeListNotFoundException : Exception
{
    public EmployeeListNotFoundException()
    {
    }

    public EmployeeListNotFoundException(string message)
        : base(message)
    {
    }

}
```

Eccezioni – Best Practices

i blocchi **catch** dovrebbero iniziare con le eccezioni più in basso nella gerarchia

- E continuare con le eccezioni più generali
- In caso contrario, si verificherà un errore di compilazione

Ogni blocco **catch** dovrebbe gestire solo queste eccezioni che si aspetta:

- Se un metodo non è in grado di gestire un'eccezione, dovrebbe essere lasciato non gestito
- Gestire tutte le eccezioni ignorando il loro tipo è una pratica non corretta (anti-pattern)

Eccezioni – Best Practices

Quando si solleva un'eccezione, passare sempre al costruttore un messaggio chiaro.

Quando si lancia un'eccezione, passare sempre una buona descrizione del problema:

Il messaggio dell' eccezione dovrebbe spiegare cosa causa il problema e come risolverlo:

Corretto: "La dimensione deve essere un numero intero nell'intervallo [1… 15]"
Corretto: "Stato non valido. Prima chiamata Initialize()"
Non corretto: "Errore imprevisto"
Non corretto: "Argomento non valido"

Eccezioni – Best Practices

Le eccezioni possono ridurre le prestazioni dell'applicazione:

- Si consiglia di generare eccezioni solo in situazioni davvero eccezionali che dovrebbero essere gestite

Il **CLR** potrebbe generare eccezioni in qualsiasi momento senza alcun modo per prevederle

- Per esempio **System.OutOfMemoryException**

Domande & approfondimenti

Academy .NET