**Gestione delle Eccezioni**

**http://www.morpheusweb.it/html/manuali/csharp/csharp\_try\_catch\_finally.asp**

Un'eccezione è un comportamento indesiderato e non previsto, che fa andare in errore un programma.Senza la gestione delle eccezioni, quando un programma esegue un'istruzione che porta ad un errore a runtime, viene interrotta e viene mostrato un messaggio di errore. Per evitare ciò possiamo utilizzare la gestione delle eccezioni.

Possiamo includere il blocco di codice che pensiamo possa portare ad un'eccezione, in un blocco try catch finally

Esempio

Consideriamo il seguente codice:

# public static void Main()

# {

# orgArr = new int[] {4, 5, 7, 9};

# for (int i = 0; i <= orgArr.GetLength(0); i++)

# Console.WriteLine("{0} ", orgArr[i]);

# }

Da come output:

Eccezione non gestita: System.IndexOutOfRangeException: Indice oltre i limiti de

lla matrice.

in ConsoleApplication1.Program.Main(String[] args) in c:\Users\roberto.venezi

ani\Documents\Visual Studio 2012\Projects\ConsoleApplication1\ConsoleApplication

1\Program.cs:riga 17

e termina l'esecuzione anche se c'era dell'altro codice da eseguire.

**try catch finally**

Vediamo con la gestione delle eccezioni:

static void Main(string[] args)

{

int[] orgArr = new int[] {4, 5, 7, 9};

try

{

for (int i = 0; i <= orgArr.GetLength(0); i++)

Console.WriteLine("{0} ", orgArr[i]);

}

catch

{

Console.WriteLine("Errore!!!");

}

finally

{

Console.WriteLine("Fine elaborazione");

}

Console.ReadKey();

}

Output:

Errore!!!

Fine elaborazione

L’indice fuori dai limiti "solleva" un'eccezione, che viene "gestita" nel blocco catch; il programma continua l'esecuzione ed il blocco finally viene eseguito.

Da notare che il codice nel blocco finally, viene eseguito comunque, è utile quindi per far eseguire al sistema operazioni di pulizia come la chiusura di recordset o di connessioni a database, perchè siamo sicuri che verranno eseguite.

**Eccezioni: gestione dell’errore**

È possibile gestire le singole eccezioni personalizzando la gestione dell’errore in base all’errore che si è verificato indicando nella clausola catch il **tipo di eccezione** che si vuole gestire; ad esempio:

:

static void Main(string[] args)

{

int[] orgArr = new int[] { 4, 5, 7, 9 };

try

{

for (int i = 0; i <= orgArr.GetLength(0); i++)

Console.WriteLine("{0} ", orgArr[i]);

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Errore!!! Superati i limiti dell'array!!!");

}

catch

{

Console.WriteLine("Errore!!!");

}

finally

{

Console.WriteLine("Fine elaborazione");

}

Console.ReadKey();

}

Output:

Errore!!! Superati i limiti dell'array!!!

Fine elaborazione

Nel costrutto try-catch è necessario specificare almeno una clausola catch oppure una finally; si possono indicare più catch, una per ogni tipo di errore che si vuole gestire, mentre è consentito inserire solo una clausola finally.

**Eccezioni: struttura Exception**

Fornisce informazioni sull’errore che si è verificato. Nell’esempio seguente la proprietà ex.Message restituisce una stringa di descrizione della causa dell'eccezione, la ex.Source il modulo nel quale si è verificato l’errore.

catch (Exception ex)

{

Console.WriteLine("Errore!!!");

Console.WriteLine("Source: " + ex.Source);

Console.WriteLine("Message: " + ex.Message);

Console.WriteLine("ToString: " + ex.ToString());

}

Da come output:

Errore!!!

Source: ConsoleApplication1

Message: Indice oltre i limiti della matrice.

ToString: System.IndexOutOfRangeException: Indice oltre i limiti della matrice.

in ConsoleApplication1.Program.Main(String[] args) in c:\Users\roberto.venezi

ani\Documents\Visual Studio 2012\Projects\ConsoleApplication1\ConsoleApplication

1\Program.cs:riga 17

Fine elaborazione

**Forzare le Eccezioni: throw**

L'istruzione throw viene utilizzata per segnalare la presenza di una situazione anomala, denominata eccezione, durante l'esecuzione di un programma. Un'istruzione throw può essere utilizzata per generare un'eccezione sia standard che creata dall’utente.

Eccezione standard:

static void Main(string[] args)

{

int[] orgArr = new int[] {4, 5, 7, 9};

try

{

for (int i = 0; i < orgArr.GetLength(0); i++)

Console.WriteLine("{0} ", orgArr[i]);

//Viene generata un'eccezione di errore nell'indice

throw new IndexOutOfRangeException();

//Viene generata una nuova eccezione di errore

throw new Exception("MioErrore");

}

catch (Exception ex)

{

Console.WriteLine("Errore!!!");

Console.WriteLine("Source: " + ex.Source);

Console.WriteLine("Message: " + ex.Message);

Console.WriteLine("ToString: " + ex.ToString());

}

finally

{

Console.WriteLine("Fine elaborazione");

}

Console.ReadKey();

}

Output:

Errore!!!

Source: ConsoleApplication1

Message: Indice oltre i limiti della matrice.

ToString: System.IndexOutOfRangeException: Indice oltre i limiti della matrice.

in ConsoleApplication1.Program.Main(String[] args) in c:\Users\roberto.venezi

ani\Documents\Visual Studio 2012\Projects\ConsoleApplication1\ConsoleApplication

1\Program.cs:riga 19

Fine elaborazione

Messaggio di errore anche se il programma è corretto!!!

Questo è dovuto all’esecuzione del comando: throw new IndexOutOfRangeException();

Eccezione non standard:

Con throw new Exception("MioErrore"); viene invece generata un’eccezione non standard.

Output:

Errore!!!

Source: ConsoleApplication1

Message: MioErrore

ToString: System.Exception: MioErrore

in ConsoleApplication1.Program.Main(String[] args) in c:\Users\roberto.venezi

ani\Documents\Visual Studio 2012\Projects\ConsoleApplication1\ConsoleApplication

1\Program.cs:riga 21

Fine elaborazione

**Eccezioni, gestirle o propagarle? throw**

Quando abbiamo una eccezione all'interno di un metodo, possiamo decidere se gestirla localmente oppure se propagarla al chiamante. Questo può essere utile se vogliamo gestire tutte le eccezioni in un blocco di codice e non per singola funzione. Nell’esempio seguente l'eccezione viene sollevata nel metodo chiamato, ma poi viene propagata e gestita nel Main.

# using System;

# namespace Sample

# {

# public class Sample

# {

# public static void Main()

# {

# string strNumero = "";

# int numero = 0;

# int divisore = 0;

# int risultato = 0;

# try

# {

# //leggo il numero

# Console.Write("Numero: ");

# strNumero = Console.ReadLine();

# numero = converti(strNumero);

# //leggo il divisore

# Console.Write("Divisore: ");

# strNumero = Console.ReadLine();

# divisore = converti(strNumero);

# //calcolo il risultato

# risultato = dividi(numero, divisore);

# Console.WriteLine("Risultato: " + risultato);

# }

# catch (Exception ex)

# {

# if (ex.Message =="Conversione")

# Console.WriteLine("Errore di conversione");

# if (ex.Message =="Divisione")

# Console.WriteLine("Divisione per zero");

# }

# }

# private static int converti(string strNumero)

# {

# try

# {

# return Convert.ToInt32(strNumero);

# }

# catch

# {

# throw new Exception("Conversione");

# }

# }

# private static int dividi(int numero, int divisore)

# {

# try

# {

# return (numero / divisore);

# }

# catch

# {

# throw new Exception("Divisione");

# }

# }

# }

# }

Output:

# Prima esecuzione

# Numero: 4

# Divisore: 2

# Risultato: 2

# Seconda esecuzione

# Numero: 4

# Divisore: e

# Errore di conversione

# Terza esecuzione

# Numero: 4

# Divisore: 0

# Divisione per zero