Le eccezioni

Fino a questo punto del corso non abbiamo mai parlato della **gestione degli errori** in C#. Semplificando, un errore è il verificarsi di una situazione imprevista durante l'esecuzione di un programma.
Gli errori nel mondo .NET, nella migliore tradizione Object Oriented (Java, C++, etc.) vengono rappresentati con le **eccezioni**: una intera gerarchia di classi per la gestione degli eventi indesiderati.
Le eccezioni vengono "sollevate" al verificarsi di una situazione anomala, come un errore. Si possono ignorare, con il rischio di avere delle brutte sorprese a run-time, oppure si possono **intercettare e gestire**.
Questo implica l'uso di un costrutto nato appositamente per l'intercettazione delle eccezioni: **try-catch-finally**.

```
try {
   //Codice che potrebbe sollevare una eccezione.
} catch {
   //Codice da eseguire in caso di eccezione.
} finally {
   //Codice da eseguire in ogni caso
}
```

Se il codice che viene eseguito all'interno del blocco try solleva una eccezione, questa viene catturata e viene eseguito il blocco catch. In ogni caso viene eseguito poi anche il blocco finally.
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.
Nella clausola catch si può indicare il **tipo di eccezione** che si vuole gestire; ad esempio:

```
catch (OverflowException ex)
{
   //Codice da eseguire in caso di errore di overflow.
}
catch (DivideByZeroException ex)
{
   //Codice da eseguire in caso di errore dovuto
   //ad una divisione per 0.
}
```

Il primo blocco catch viene eseguito se si verifica un errore di overflow, ad esempio quando si cerca di assegnare ad una variabile un valore al di fuori del suo range di definizione. Il secondo viene invocato se si effettua una divisione per 0.

In entrambi i casi, l'oggetto «ex» contiene varie informazioni sull'errore. Ad esempio, la proprietà **ex.Message** restituisce una stringa di descrizione della causa dell'eccezione.
Aggiorniamo ora l'applicazione che abbiamo realizzato nella lezione precedente aggiungendo la gestione degli errori. In particolare, consideriamo l'istruzione

```
picImmagine.Image = Bitmap.FromFile(txtNomeFile.Text);
```

Nel caso in cui il file specificato non esista, il metodo **FromFile** lancia un'eccezione di tipo **FileNotFoundException**. Inseriamo, quindi, tale istruzione in un blocco try…catch:

```
try {
   picImmagine.Image = Bitmap.FromFile(txtNomeFile.Text);
} catch (FileNotFoundException){
   MessageBox.Show("Impossibile trovare il file " + txtNomeFile.Text + ".",
"Errore", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Dal momento che anche le eccezioni sono classi, anch'esse **sono raggruppate all'interno di namespace**. Per esempio, perché il Framework sappia dove cercare la classe FileNotFoundException, tra le clausole «using» va aggiunta quella relativa al namespace «System.IO».
Ogni volta che, cercando di caricare un file, si verifica un'eccezione di tipo «FileNotFoundException», viene eseguito il blocco catch corrispondente, il quale visualizza una finestra di messaggio. L'errore di file non trovato, però, non è l'unico che può capitare quando si cerca di aprire un file: ad esempio, se la memoria a disposizione non è sufficiente per completare il caricamento, si ottiene un errore di tipo **OutOfMemoryException**.
Ancora, se si preme il pulsante «Visualizza» senza aver digitato alcun nome di file, si verifica una **ArgumentException**.
Aggiungiamo, quindi, un secondo blocco catch per trattare tutti gli altri casi, ottenendo complessivamente:

```
try
{
  picImmagine.Image = Bitmap.FromFile(txtNomeFile.Text);
}
catch(ileNotFoundException)
{
  MessageBox.Show(
    "Impossibile trovare il file " + txtNomeFile.Text + ".",
    "Errore",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch(Exception ex)
{
  MessageBox.Show(
    "Errore durante uil caricamento: " + ex.Message,
    "Errore",
    MessageBoxButton.OK, MessageBoxIcon.Error);
}
```

Poiché **Exception** è la classe base da cui derivano tutte le altre eccezioni, il secondo blocco catch è eseguito ogni volta che il metodo FromFile lancia un'eccezione che non sia di tipo «FileNotFoundException» (e che non derivi da essa).
In questo caso abbiamo usato l'informazione contenuta nell'eccezione «ex» per sapere con esattezza l'origine dell'errore.

È sempre meglio evitare la gestione delle eccezioni, cercando di prevedere, dove possibile, le cause che possono portare ad una situazione imprevista: un abuso di try…catch, infatti, può appesantire molto un programma. Nel nostra applicazione, ad esempio, l'errore ArgumentException può sempre essere evitato semplicemente facendo in modo che il metodo FromFile venga richiamato solo se è stato digitato qualcosa nella casella di testo.