

# Week 7




**Antonia Sacchitella**

Analyst@icubedsrl

[Antonia.sacchitella@icubed.it](mailto:Antonia.sacchitella@icubed.it)



# Week 7 - Agenda

- Exception
  - Gestione delle Exception
  - Riconoscere quando utilizzare un'eccezione
  - Creare eccezioni custom
- Object Oriented Design
  - Descrizione degli obiettivi
  - UML – Lo standard adatto per la progettazione
-  Esercitazione



# Eccezioni

Le eccezioni sono errori dati da una situazione imprevista avvenuta durante l'esecuzione di un programma.

Non sono solo date da «sviste» dello sviluppatore, ma anche da altre problematiche:

- Autorizzazione fallita
- Problema di Hardware
- Risorsa non attualmente disponibile
- ...

# Errori ed eccezioni

Durante l'esecuzione di codice è possibile che si verifichino diverse tipologie di errore.

**Errori di utilizzo.** Un errore di utilizzo rappresenta un errore nella logica del programma che può generare un'eccezione. Tuttavia, l'errore non può essere risolto tramite la gestione delle eccezioni, ma modificando il codice difettoso.

**Errori del programma.** Un errore di programma è un errore di run-time che non può essere necessariamente evitato scrivendo codice privo di bug.

**Errori di sistema.** Un errore di sistema è un errore di run-time che non può essere gestito a livello di programmazione in modo significativo. In genere, gli errori di sistema non vengono gestiti mediante la gestione delle eccezioni.

# System.Exception

Tutto in C# viene rappresentato con una classe.

Alla verifica di una eccezione, il common language runtime associa al contest di runtime un'istanza di tipo è (o deriva da) Exception.

Tale classe ha le seguenti proprietà:

Nome	Descrizione
Message	Descrizione dell'errore
Source	Nome dell'assembly che ha sollevato l'eccezione.
StackTrace	Lo stack delle chiamate nel momento in cui si verifica l'eccezione.
HelpLink	Contiene un eventuale url per rimandare ad una pagina di aiuto
TargetSite	Definizione del metodo che ha sollevato l'eccezione

# Eccezioni dal compilatore

Eccezione	Descrizione
<a href="#"><u>ArithmeticException</u></a>	Classe di base per le eccezioni che si verificano durante operazioni aritmetiche, quali <a href="#"><u>DivideByZeroException</u></a> e <a href="#"><u>OverflowException</u></a> .
<a href="#"><u>ArrayTypeMismatchException</u></a>	Generata quando una matrice non può archiviare un dato elemento perché il tipo effettivo dell'elemento non è compatibile con il tipo effettivo della matrice.
<a href="#"><u>DivideByZeroException</u></a>	Generata quando viene eseguito un tentativo di dividere un valore integrale per zero.
<a href="#"><u>IndexOutOfRangeException</u></a>	Generata quando viene eseguito un tentativo di indicizzare una matrice, quando l'indice è minore di zero o supera i limiti della matrice.
<a href="#"><u>InvalidCastException</u></a>	Generata quando una conversione esplicita dal tipo di base a un'interfaccia o a un tipo derivato ha esito negativo in fase di runtime.

# Eccezioni dal compilatore

Eccezione	Descrizione
<a href="#">NullReferenceException</a>	Generata quando viene eseguito un tentativo di fare riferimento a un oggetto il cui valore è <a href="#">null</a> .
<a href="#">OutOfMemoryException</a>	Generata quando un tentativo di allocazione della memoria tramite l'operatore <a href="#">new</a> ha esito negativo. Questa eccezione indica che la memoria disponibile per il Common Language Runtime è stata esaurita.
<a href="#">OverflowException</a>	Generata quando si verifica un overflow di un'operazione aritmetica in un contesto checked.
<a href="#">StackOverflowException</a>	Generata quando viene esaurito lo stack di esecuzione da un numero eccessivo di chiamate in sospeso verso il metodo; in genere indica un problema di ricorsione molto profondo o infinito.
<a href="#">TypeInitializationException</a>	Generata quando un costruttore statico genera un'eccezione e non è presente una clausola catch compatibile per intercettarla.

# Gestione delle exception

Quando viene generata un' eccezione, l' esecuzione del programma viene interrotta.

Viene associata al contesto di runtime il tipo di eccezione generata e ripercorrendo lo stack delle chiamate fino al Main si ricerca un **gestore** dell'errore.

Se esiste un gestore nello stack : prende in carico l' eccezione.

Se non esiste un gestore nello stack : l'eccezione comporta la chiusura dell'applicazione \*

\*in Debug si vede la riga che ha generato l'eccezione.



# Intercettare le Eccezioni

Il gestore si mette in pratica tramite il costrutto try-catch:

Try: codice che potrebbe causare un' eccezione

Catch : gestore dell' eccezione

Finally: blocco di codice da eseguire sia nel caso che vada o non vada in errore

```
try
{
    sr = new StreamReader(@"file-inesistente.txt");
    string content = sr.ReadToEnd();
}
catch (FileNotFoundException ex)
{
    Console.WriteLine("Errore di I/O:\r\n{0}", ex);
}
finally
{
    if (sr != null)
        sr.Close();
}
```

# Intercettare le Eccezioni

In presenza di istruzioni che possono provocare più di una tipologia di eccezioni, è necessario inserire i blocchi di intercettazione da quello più specifico al meno specifico.

Questo permette di avere una gestione più puntuale delle varie problematiche che si possono presentare.

```
SqlConnection conn =  
    new SqlConnection(strConn);  
  
try {  
    conn.Open();  
    ElaboraRisultati(conn);  
} catch (SQLException exc) {  
    // informazioni specifiche di SQLException  
} catch (Exception ex) {  
    // qui entra solo se non è una SQLException  
} finally {  
    // questo codice viene sempre eseguito  
    conn.Close();  
}
```

# Best Practices

- Quando si sta lanciando un'eccezione dovuta da un errore di utilizzo, ovvero da un errore nella logica del programma eseguita dallo sviluppatore che chiama il metodo. Ad esempio `ArgumentException`, `ArgumentNullException`, `InvalidOperationException` oppure `NotSupportedException`.
- La stringa fornita al costruttore dell'oggetto `Exception` quando si crea un'istanza dell'oggetto eccezione dovrebbe descrivere l'errore in modo che lo sviluppatore possa correggerlo.
- È consigliabile generare l'eccezione più derivata possibile. Se, ad esempio, un metodo richiede che un argomento sia un membro valido di un tipo di enumerazione, è necessario generare un'eccezione `InvalidEnumArgumentException` (la classe più derivata) invece di un oggetto `ArgumentException`.

# Demo



# Esercitazione

Scrivere un programma che istanzia un vettore di interi di 5 posizioni chieda all'utilizzatore di inserire un numero in una posizione specificata.

Stampare su file l'array finale.

Riportare una duplice versione gestendo l'eventualità che vengano violati i limiti dell'array sia attraverso un controllo preventivo che attraverso l'utilizzo delle eccezioni.



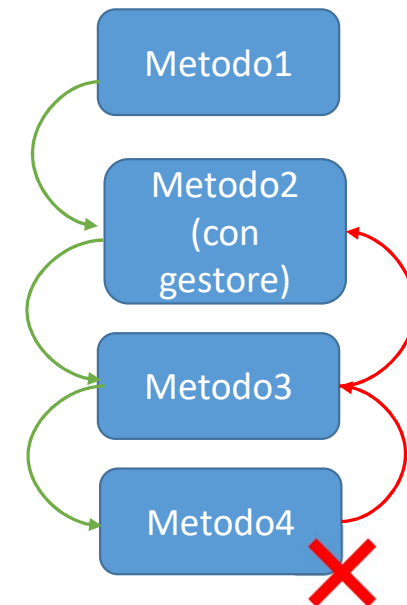
# Gestore - Catch

Quando viene generata l'eccezione, essa viene propagata da metodo a metodo risalendo lo stack delle chiamate finché non trova il gestore.

Non appena viene trovato il gestore, il sistema di "notifica" viene interrotto.

L'eccezione viene marcata come gestita.

Si esegue il blocco di codice specificato nel gestore.



# Exception throwing



Solitamente le eccezioni vengono 'sollevate' dal runtime .NET quando si verifica una condizione di errore.

È possibile anche che una applicazioni 'sollevi' una eccezione in maniera autonoma tramite la keyword throw.

```
public string Description(int value) {  
    if(value < 0)  
        throw new ArgumentException();  
    // ...  
}
```

# Exception throwing



Solitamente le eccezioni vengono 'sollevate' dal runtime .NET quando si verifica una condizione di errore.

È possibile anche che una applicazioni 'sollevi' una eccezione in maniera autonoma tramite la keyword throw.

```
public string Description(int value) {  
    // ...  
    public string Description(int value)  
    {  
        try  
        {  
            // ...  
        }  
        catch(Exception ex)  
        {  
            throw ex;  
        }  
    }  
}
```

Reset the Stack Trace



# Exception throwing



Solitamente le eccezioni vengono 'sollevate' dal runtime .NET quando si verifica una condizione di errore.

È possibile anche che una applicazioni 'sollevi' una eccezione in maniera autonoma tramite la keyword throw.

```
public string Description(int value) {  
    // ...  
    try  
    {  
        //  
    }  
    catch(  
    {  
        thn  
    }  
}  
  
    try  
    {  
        public string Description(int value)  
        {  
            try  
            {  
                // ...  
            }  
            catch(Exception ex)  
            {  
                throw;  
            }  
        }  
    }  
}
```

**Maintain the Stack Trace**

# Demo

Exception throwing



# Custom Exceptions



È possibile definire delle eccezioni custom semplicemente estendendo la classe base di tutte le eccezioni, `Exception`.

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

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

    public EmployeeListNotFoundException(string message, Exception inner)
        : base(message, inner)
    {
    }
}
```

# Quando NON sollevare Eccezioni

Non sempre è il caso si sollevare un' eccezione.

Non usare eccezioni per modificare il flusso di un programma come parte dell'esecuzione normale. Utilizzare le eccezioni per segnalare e gestire le condizioni di errore.

Le eccezioni non devono essere restituite come valore restituito o parametro anziché essere generate.

Non generare `System.Exception` , `System.SystemException` , `System.NullReferenceException` o `System.IndexOutOfRangeException` intenzionalmente dal codice sorgente.

# Demo



# Esercitazione

Scrivere un programma di gestione degli utenti.

Ogni utente ha uno username e una password.

Creare in un database gestito con ADO una tabella contenente tutte le informazioni degli utenti.

Creare un'eccezione specifica che possa essere lanciata quando il login fallisce.

