

# Corso di Programmazione in Visual Studio

**Davide Maggiulli**

Davide.maggiulli@gmail.com



# Agenda

- Introduzione a .NET
- Visual Studio
- Linguaggio VB.NET
- OOP
- ADO.NET



# .NET Framework



# .NET Framework – Che cos'è?

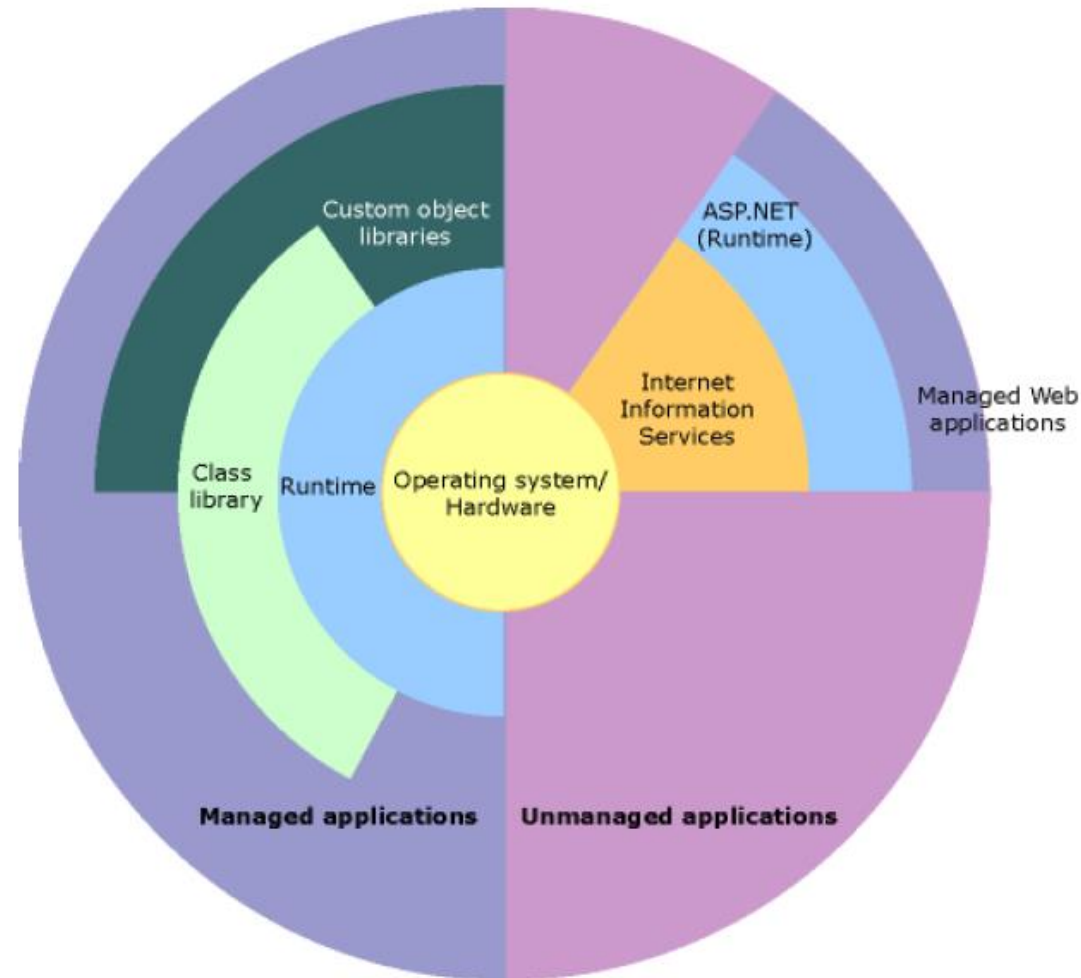
- Un componente di Windows che permette di sviluppare, eseguire e distribuire applicazioni e servizi web.
- Obiettivi:
  - Fornire un unico ambiente di sviluppo object-oriented sia per applicazioni eseguite localmente che in remoto
  - Mettere a disposizione un ambiente di esecuzione dei programmi che riduca problematiche di deployment e conflitti fra versioni diverse
  - Aumentare la sicurezza e affidabilità del codice
  - Fornire agli sviluppatori strumenti analoghi in applicazioni Windows, Web, Windows Phone.

# .NET Framework struttura

- Si compone di due elementi principali: CLR e Class Library.
- Common Language Runtime (CLR)
  - Si occupa dell'esecuzione dei programmi
  - Fornisce servizi base quali gestione della memoria e degli thread
  - È responsabile della sicurezza e affidabilità dei programmi
  - I programmi eseguiti dal CLR sono detti “managed applications”
- Class Library
  - Una vasta collezione, gerarchica ed estendibile, di classi
  - Sia funzionalità di base (file, stringhe, strutture dati, accesso a database), che per specifiche tipologie di applicazioni (Console applications, Windows GUI applications, Web services, ...)

# .NET Framework – Esecuzione delle applicazioni

- Managed applications: programmi eseguiti dal CLR
- Unmanaged applications: applicazioni “tradizionali”
  - Eseguite direttamente dal S.O.
  - Es. DBMS, web-server
  - possono “ospitare” al loro interno il .NET Framework, chiedendo al CLR di eseguire “componenti managed”



# CLR e CLI: non solo Windows

- CLR è l'implementazione Microsoft di CLI (Common Language Infrastructure)
  - CLI è uno standard ISO (ISO/IEC 23271:2003)
- Esistono già altre implementazioni di CLI:
  - SSCLI (Shared Source Common Language Infrastructure): disponibile per Windows, FreeBSD e Macintosh
  - .NET Compact Framework: per dispositivi PocketPC, SmartPhone, ...
  - Mono: implementazione Open Source per Linux
  - ...

# CLR – Terminologia

- CTS - Common Type System:
  - sistema di tipi unificato e inter-linguaggio
  - Due categorie di tipi (Value Type e Reference Type)
- CLS - Common Language Specification
  - Uno standard a cui qualsiasi linguaggio per .NET deve aderire; prevede un sottoinsieme minimo del CTS (utile per garantire interoperabilità fra linguaggi differenti)
  - In questo modo tutti i linguaggi .NET possono beneficiare del Class Library
- CIL - Common Intermediate Language (MSIL nell'implementazione Microsoft)
  - Un linguaggio indipendente dalla CPU che può essere efficientemente tradotto nel linguaggio macchina di una data CPU
- JIT- Just In Time Compiler
  - Non tutto il codice CIL di un programma viene sempre eseguito: solo la parte necessaria viene compilata un istante prima della sua esecuzione
  - Il codice compilato viene memorizzato per successive esecuzioni
- VES – Virtual Execution System: l'ambiente di esecuzione (macchina virtuale)

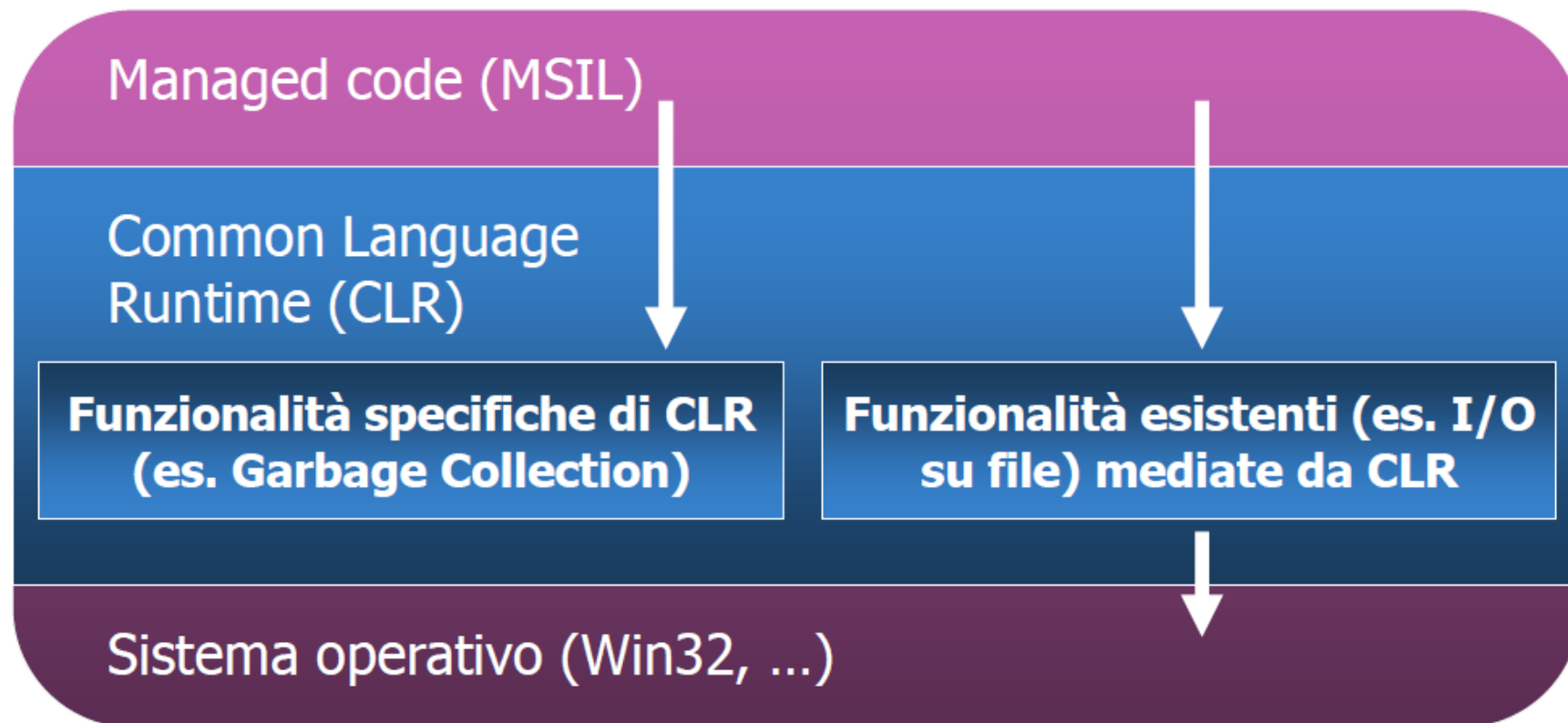


# CLR - Terminologia

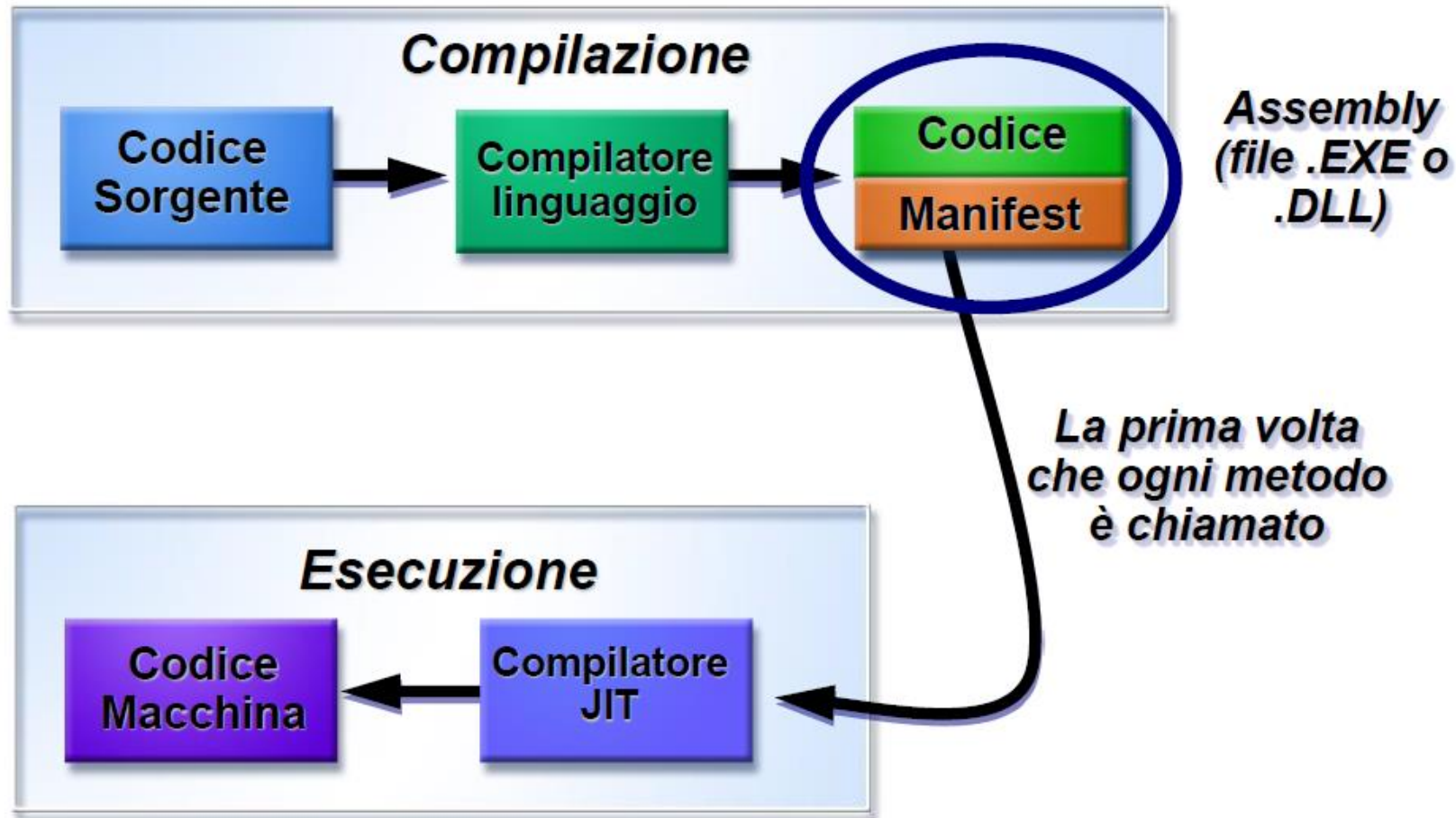
- Assembly
  - Insieme di funzionalità sviluppate e distribuite come una singola unità applicativa, composta da uno o più file
  - Completamente auto-descrittivo grazie al suo manifest
- Manifest
  - Stabilisce l'identità dell'assembly in termini di nome, versione, livello di condivisione tra applicazioni diverse, firma digitale, ...
  - Definisce quali file costituiscono l'implementazione dell'assembly
  - Specifica le dipendenze in fase di compilazione da altri assembly
  - ...
- Application Domain
  - Unità di elaborazione .NET (un assembly deve essere caricato in un Application Domain per poter essere eseguito)
  - Più "leggero" di un processo (più Application Domain possono risiedere nello stesso processo, ma vi sono meccanismi di sicurezza e isolamento)

# CLR – Esecuzione managed applications

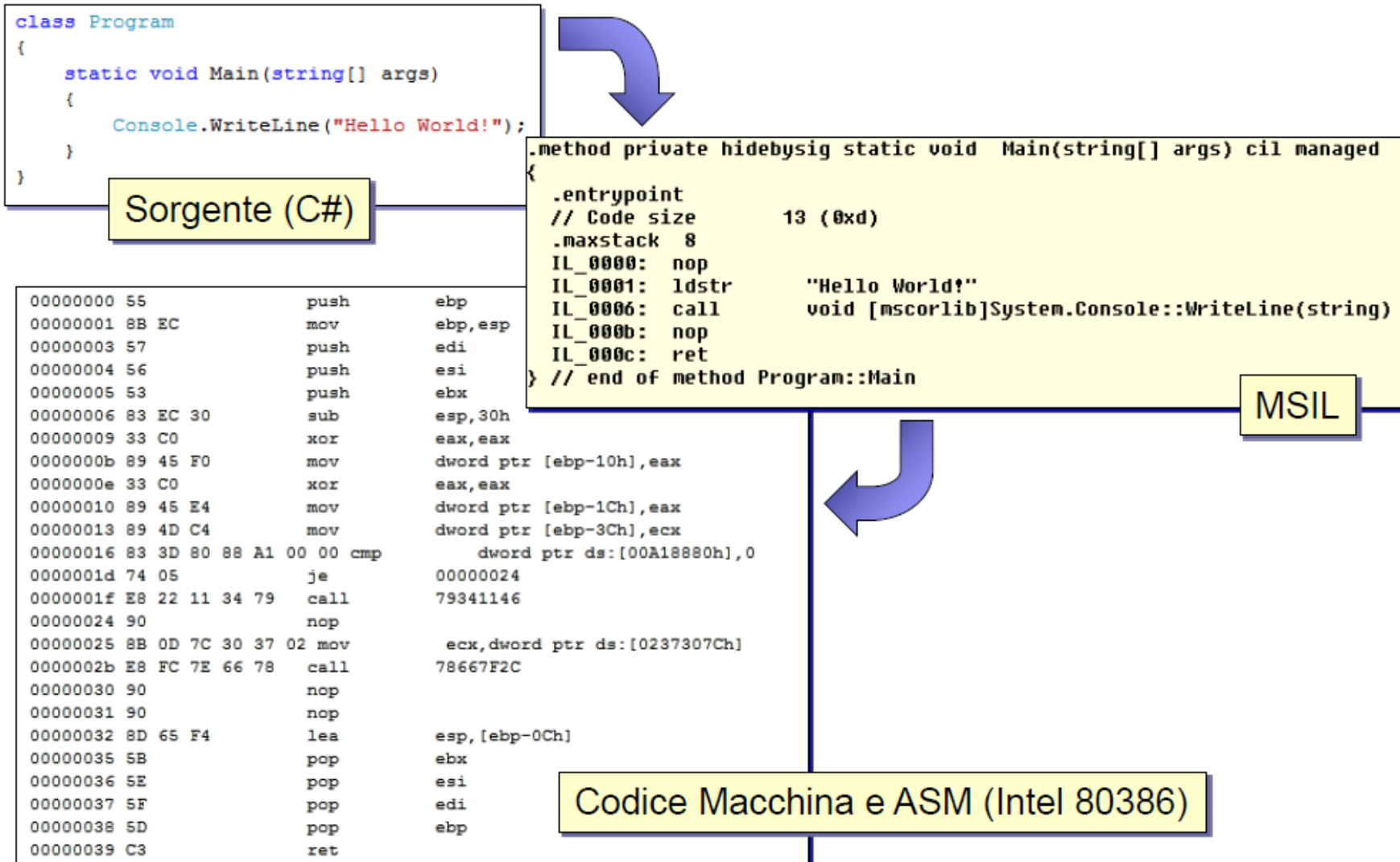
- Le managed applications sono scritte in MSIL, che il CLR è in grado di eseguire, offrendo vari servizi



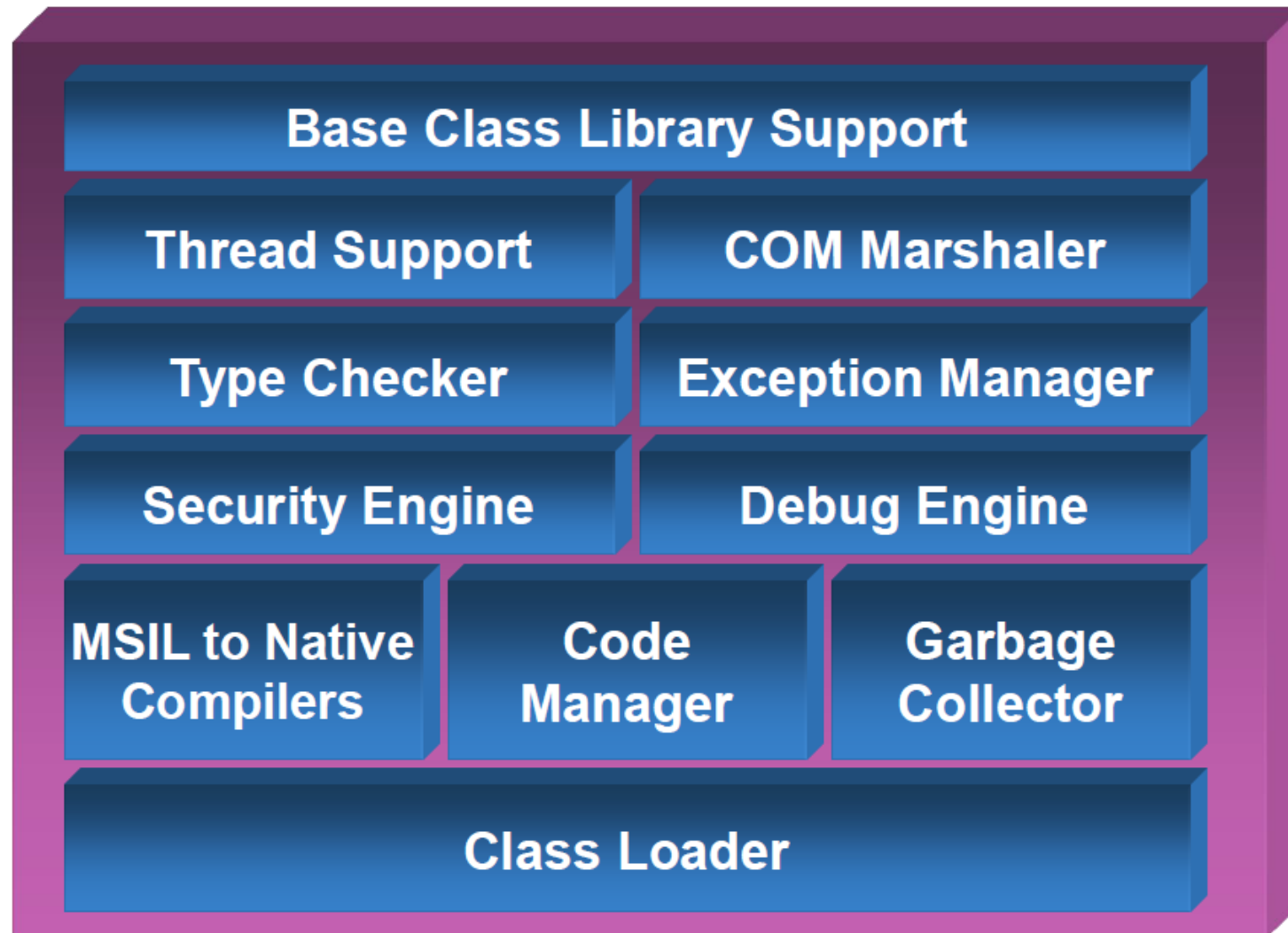
# CLR, codice MSIL e compilatore JIT



# Un esempio: sorgente – MSIL - ASM



# CLR - Struttura



# CLR – Vantaggi

- Ambiente object-oriented
  - Qualsiasi entità è un oggetto
  - Classi ed ereditarietà pienamente supportati (anche tra linguaggi diversi)
- Riduzione errori comuni di programmazione
  - Linguaggi fortemente tipizzati
  - Gestione eccezioni
  - Prevenzione dei memory leak: Garbage Collection
- Indipendenza dal sistema operativo
  - Senza perdere troppa efficienza grazie al JIT che può ottimizzare il codice per la specifica piattaforma
- Piattaforma multi-linguaggio
  - I componenti di un'applicazione possono essere scritti con linguaggi diversi

# Class Library



# Class Library – Classi di base

- Tipi di dati, conversioni, formattazione
- Strutture dati: Array, Liste, Hash, ...
- I/O: file di testo e binari, compressione, ...
- Rete: HTTP, TCP/IP socket, ...
- Sicurezza: Permessi, crittografia, ...
- Testo: Codifiche, espressioni regolari, ...
- Supporto per la localizzazione (multi-lingua)
- ...



# Linguaggi per .NET

- Qualsiasi linguaggio conforme al CLS
- Forniti da Microsoft
- C++, C#, F#, VB.NET, Jscript
- Forniti da terze parti
- Perl, Python, Pascal, APL, COBOL, Eiffel, Haskell, ML, Oberon, Scheme, Smalltalk, ...
- Tutti i linguaggi .NET possono utilizzare la Class Library e le funzionalità del framework, ma il linguaggio “principe” è il C#!

# Linguaggi per .NET – Esempi

```
Class HelloWorldApp
  Shared Sub Main()
    System.Console.WriteLine("Hello, world!")
  End Sub
End Class
```

Visual Basic .NET

```
class HelloWorldApp
{ static void Main()
{
    System.Console.WriteLine("Hello, world!");
}
}
```

C#

```
000330 IDENTIFICATION DIVISION.
000340 PROGRAM-ID. MAIN.
000350
000360 ENVIRONMENT DIVISION.
000370
000380 DATA DIVISION.
000390 WORKING-STORAGE SECTION.
000400
000410 PROCEDURE DIVISION.
000420     DISPLAY "Hello, World!"
000430 END PROGRAM MAIN.
```

COBOL.NET

.NET Core

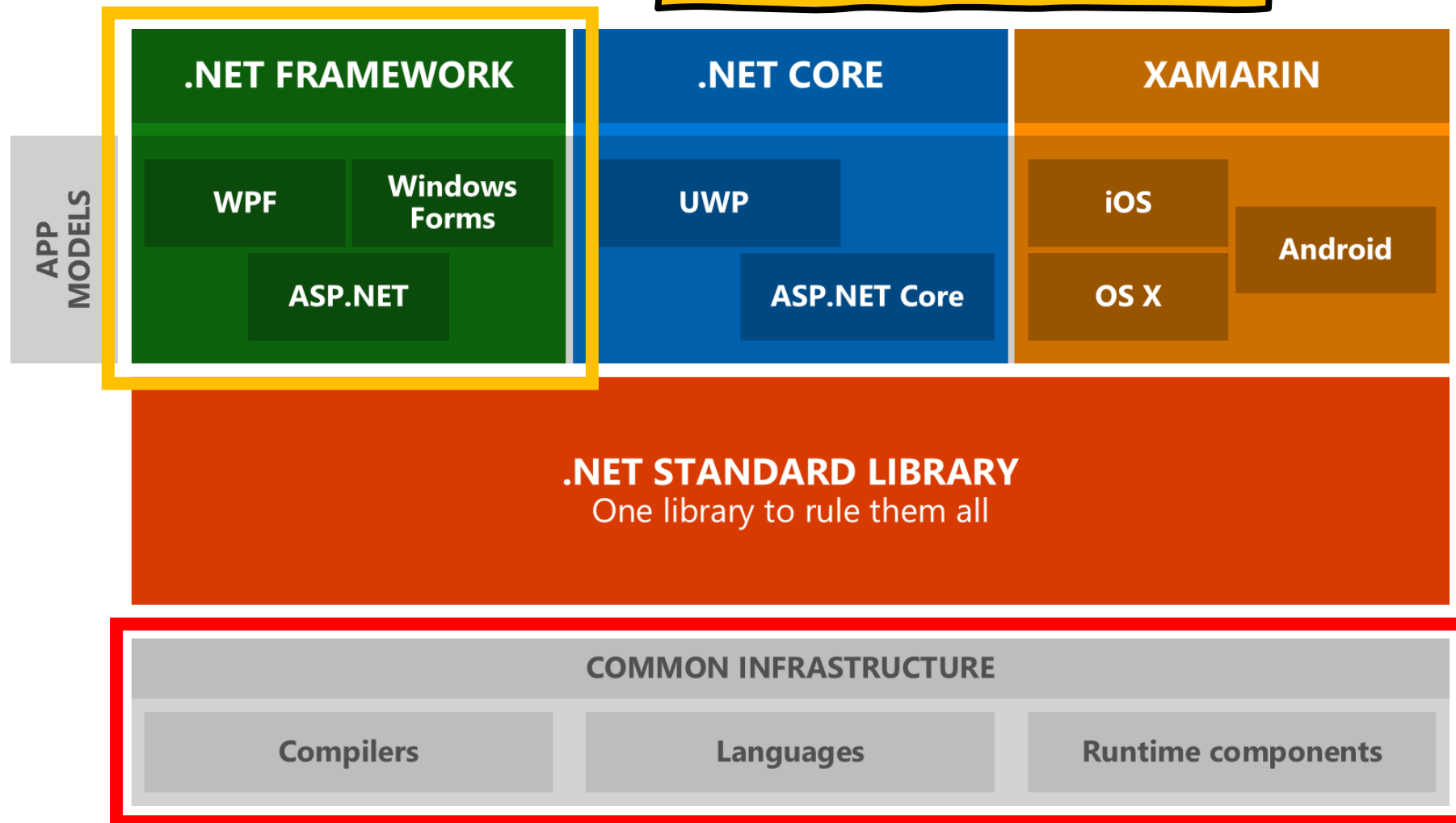


# .NET

## .NET Framework

- First Release: 2001 / 2002
- Runs on Windows ONLY
- Use for Legacy app

.NET



# .NET

- .NET Framework
  - First release: 2001/2002
  - Gira solo su Windows
  - Legacy apps
- .NET Core
  - First release: 2015
  - Cross-platform (Win, MacOS, Linux)
  - Open Source
  - Per tutti i nuovi sviluppi
- NET Standard
  - specifica un insieme di API che le piattaforme .NET devono implementare per indicare il livello di compatibilità
  - È solo uno standard (come HTML5)
- Mono/Xamarin
  - First release: 2004
  - Reimplementazione open-source del .NET Framework
  - Sviluppo per app mobile o Unity (games)
- BCL: Base Class Library
  - libreria di classi base, utilizzabile per realizzare i programmi nei diversi linguaggi
  - Garbage Collector – manutenzione della memoria (al contrario dei linguaggi unmanaged, dove tutto deve essere gestito dal programmatore).

# .NET Standard Library

Ogni versione è retro-compatibile con le precedenti

1.6: .NET Core 1

2.0: .NET Core 2

.NET Standard	1.01.0	1.11.1	1.21.2	1.31.3	1.41.4	1.51.5	1.61.6	2.02.0
.NET Core	1.01.0	1.01.0	1.01.0	1.01.0	1.01.0	1.01.0	1.01.0	2.02.0
.NET Framework (con .NET Core 1.x SDK))	4.5	4.5	4.5.1	4.64.6	4.6.1	4.6.2		
.NET Framework (con .NET Core 2.0 SDK)	4.5	4.5	4.5.1	4.64.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.64.6	4.64.6	4.64.6	4.64.6	4.64.6	4.64.6	4.64.6	5.45.4
Xamarin.iOS	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.14
Xamarin.Mac	3.03.0	3.03.0	3.03.0	3.03.0	3.03.0	3.03.0	3.03.0	3.83.8
Xamarin.Android	7.07.0	7.07.0	7.07.0	7.07.0	7.07.0	7.07.0	7.07.0	8.08.0
UWP	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.0.1629910.0.16299	10.0.1629910.0.16299	10.0.1629910.0.16299
Windows	8.08.0	8.08.0	8.18.1					
Windows Phone	8.18.1	8.18.1	8.18.1					
Silverlight per Windows Phone	8.08.0							



# VB.NET

- Value types VS Reference Types
- Classi e interfacce (classi astratte, enum)
- Incapsulamento, Ereditarietà, Polimorfismo
- Eccezioni
- Gestione degli eventi, delegates
- Lettura e Scrittura su File
- Multithreading
- LINQ e Lambda



# Value Type

Contengono direttamente il dato nell'ambito dello stack del thread.

Una copia di un Value Type implica la copia dei dati in esso contenuti.

Le modifiche hanno effetto solo sull'istanza corrente.

Contengono sempre un valore (null **non** è direttamente ammesso).

I Value Type comprendono:

- i tipi primitivi come int, byte, bool, ecc.
- **enum**, **struct** (definiti dall'utente).

```
Dim i as Integer = 16
Dim b as Boolean = true
Dim d as Double d = 0d;
Dim a As DateTime = DateTime.Now
```





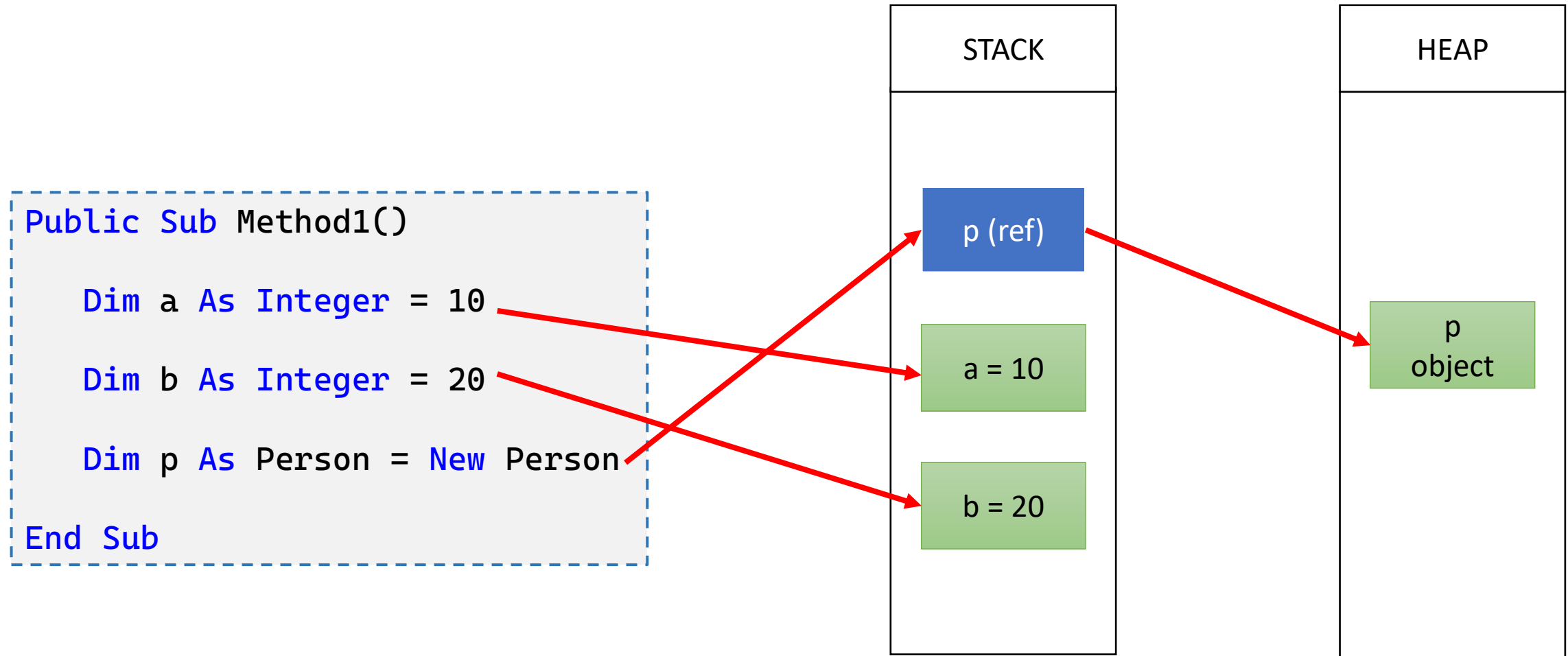
# Reference Type

- Contengono solo un riferimento ad un oggetto nell'ambito dell'heap
- La copia di un Reference Type implica la duplicazione del solo reference
- Le modifiche su due reference modificano l'oggetto a cui puntano
- Il reference che non referencia nessuna istanza vale **null**
- **Tutte le classi sono Reference Type!**

```
//Attenzione: il tipo string è un caso particolare perché è immutabile.  
Dim s As String = "VB.NET"  
Dim ds As DataSet = New DataSet()  
Dim p As Person = New Person()
```



# Value Type & Reference Type



# Enumerazioni

Un **enum** è una "classe" speciale che rappresenta un gruppo di costanti (variabili non modificabili / di sola lettura).

```
Public Enum TimeOfDay  
    Morning = 0  
    Afternoon = 1  
    Evening = 2  
End Enum
```

Possiamo **accedere ad un valore** utilizzando:

```
Dim timeOfDay As TimeOfDay = TimeOfDay.Afternoon
```



# Classi

Una classe è come un costruttore di oggetti o un "blueprint" per la creazione di oggetti.

```
Public Class Person  
    //...  
End Class
```

Una classe può contenere ed eventualmente esporre una sua interfaccia:

- Dati (**campi** e **proprietà**)
- Funzioni (**metodi**)



# Classi, campi e proprietà

## Campo

```
Public Class Person
    Public Name As String
End Class

Dim p As Person = New Person()
p.Name = "Mario Rossi"
Console.WriteLine(p.Name)
```



# Classi e proprietà

## Proprietà

```
Public Class Person
    Private _name As String

    Public Property Name As String
        Get
            Return _name
        End Get
        Set(value As String)
            _name = value
        End Set
    End Property
End Class

Dim p As Person = New Person()
p.Name = "Mario Rossi"
Console.WriteLine(p.Name)
```

# Classi e proprietà



## Proprietà read-only

```
Public ReadOnly Property Name As String
    Get
        Return _name
    End Get
End Property

Dim p As Person = New Person()
p.Name = "Mario Rossi"
Console.WriteLine(p.Name)
```

## Proprietà read-only

```
Public WriteOnly Property Name As String
    Set(ByVal value As String)
        _name = value
    End Set
End Property

Dim p As Person = New Person()
p.Name = "Mario Rossi"
Console.WriteLine(p.Name)
```



# Classi e proprietà

- È il modo migliore per soddisfare uno dei pilastri della programmazione OOP: *incapsulamento*
- Una proprietà può provvedere accessibilità in lettura (**get**) scrittura (**set**) o entrambi
- Si può usare una proprietà per ritornare valori calcolati o eseguire una validazione





# Metodi di una classe

- In sostanza la dichiarazione di un metodo è composta di:
  - zero o più keyword
  - il tipo di ritorno del metodo oppure **void** nel caso di sub procedure
  - il nome del metodo
  - l'elenco dei parametri tra parentesi tonde
- La *firma* (*signature*) di un metodo è rappresentata dal nome, dal numero dei parametri e dal loro tipo; il valore ritornato **non** fa parte della firma.

```
Public Sub MethodA(ByVal inString  
As String)  
    'Statements  
End Sub
```

```
Public Function MethodB(ByRef a As  
Integer) As Integer  
    Return 2  
End Function
```



# Accessibilità

- I tipi definiti dall'utente (classi, strutture, enum) e i membri di classi e strutture (campi, proprietà e metodi) possono avere accessibilità diversa (*accessor modifier*):
  - **public** Accessibile da tutte le classi
  - **protected** Accessibile solo dalle classi derivate
  - **private** Non accessibile dall'esterno
  - **Friend** Accessibile all'interno dell'assembly
  - **Protected Friend** Combinazione delle due
- Differenziare l'accessibilità di un membro è fondamentale per realizzare l'*incapsulamento*.
- L'insieme dei membri esposti da una classe rappresenta la sua *interfaccia*.



# Ereditarietà

- Si applica quando tra due classi esiste una relazione “è un tipo di”. Esempio: **Customer** è un tipo di **Person**.
- Consente di specializzare e/o estendere una classe.
- Si chiama *ereditarietà* perché la classe che deriva (**classe derivata**) può usare tutti i membri della classe ereditata (**classe base** – keyword **MyBase**) come se fossero propri, ad eccezione di quelli dichiarati privati.

```
Public Class Person
    Private _name As String

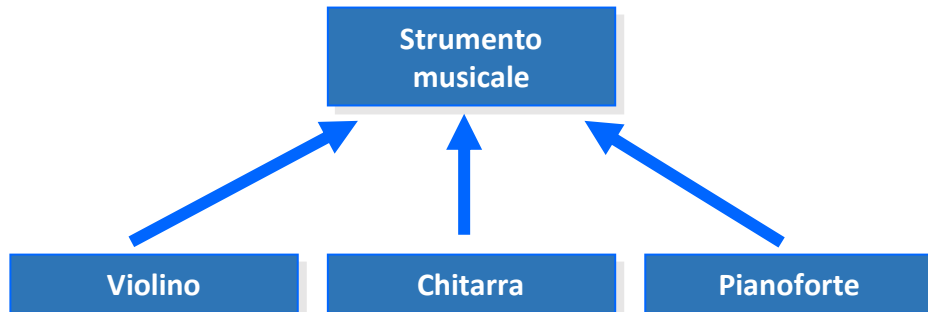
    Public Property Name As String
        Get
            Return _name
        End Get
        Set(value As String)
            _name = value
        End Set
    End Property
End Class
```

```
Public Class Customer
    Inherits Person

End Class
```

# Polimorfismo

- Il *polimorfismo* è la possibilità di trattare un'istanza di un tipo come se fosse un'istanza di un altro tipo.
- Il polimorfismo è subordinato all'esistenza di una relazione di derivazione tra i due tipi.
- Affinchè un metodo possa essere polimorfico, deve essere marcato come **Overridable** o **MustInherit**.



```
Public Class Strumento
    Public Overridable Sub Accorda()

    End Sub
End Class

Public Class Violino
    Inherits Strumento

    Public Overrides Sub Accorda()
        MyBase.Accorda()
    End Sub
End Class

Public Class Orchestra
    Public violino As Strumento
    Public chitarra, pianoforte As Strumento

    Public Sub New()
        violino = New Violino
        violino.Accorda()
    End Sub
End Class
```

# Interfacce

- Un'interfaccia definisce un contratto che la classe che la implementa deve rispettare
- Un'interfaccia è priva di qualsiasi implementazione e di modificatore di accessibilità (**public**, **private**, ecc.)
- Una classe può implementare più interfacce contemporaneamente.

```
Public Interface IStrumento
    Sub Accorda()
    Sub StartTime()
End Interface
```

```
Public Class Pianoforte
    Implements IStrumento

    Public Sub Accorda() Implements
        IStrumento.Accorda

    End Sub

    Public Sub StartTime() Implements
        IStrumento.StartTime

    End Sub
End Class
```



# Gestione delle eccezioni

- **try** serve a racchiudere gli statement per i quali si vogliono intercettare gli errori (chiamate annidate comprese).
- **catch** serve per catturare uno specifico errore. Maggiore è la indicazione dell'eccezione, maggiore è la possibilità di recuperare l'errore in modo soft.
- **finally** serve ad indicare lo statement finale da eseguire sempre, sia in caso di errore, sia in caso di normale esecuzione.

```
Dim conn As SqlConnection = New SqlConnection(cs)

Try
    conn.Open()
    ElaborarResultati(conn)
Catch exc As SQLException
    'informazioni specifiche di SQLException
Catch ex As Exception
    'qui entra solo se non è una SQLException
Finally
    'questo codice viene sempre eseguito
    conn.Close()
End Try
```



# Delegate

- I **delegate** sono l'equivalente .NET dei puntatori a funzione del C/C++ unmanaged, ma hanno il grosso vantaggio di essere tipizzati
- In C# lo si dichiara con la parola chiave **delegate**

```
Public Delegate Sub MyDelegate(i As Integer)
```

- Il compilatore crea di conseguenza una classe che deriva da **System.Delegate** oppure **System.MulticastDelegate** (di nome **MyDelegate**)
- Queste due classi sono speciali e solo il compilatore può derivarle



# Delegate

- Da programma il delegate viene istanziato passandogli nel costruttore il nome del metodo di cui si vuole creare il delegate.

```
Dim del As MyDelegate = AddressOf MyMethod
```



```
Public Sub MyMethod(num As Integer)  
End Sub
```

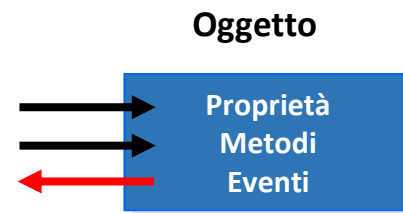
- L'istanza può finalmente essere invocata

```
del(5) // esegue MyMethod (integer)
```



# Eventi

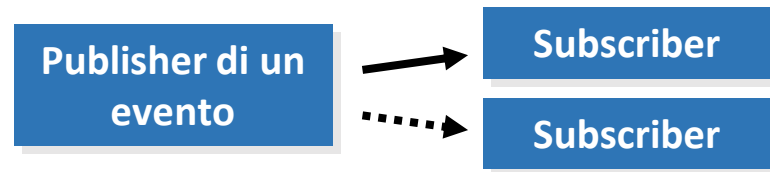
- Un **evento** è un membro che permette alla classe di inviare notifiche verso l'esterno
- L'evento mantiene una lista di *subscriber* che vengono iterati per eseguire la notifica
- Tipicamente sono usati per gestire nelle Windows Forms le notifiche dai controlli all'oggetto container (la Form)



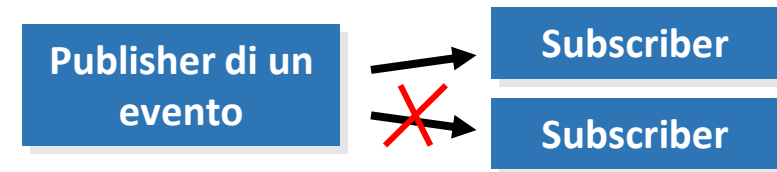
- Si parla di:
  - *Publisher* Inoltra gli eventi a tutti i subscriber
  - *Subscriber* Riceve gli eventi dal publisher

# Eventi

- Ciascun subscriber deve essere aggiunto alla lista del publisher (*subscribe*) oppure rimosso (*unsubscribe*).



```
c.MyEvent += f;
```



```
c.MyEvent -= f;
```



# Accesso ai File

La classe `File` fornisce metodi statici per la maggior parte delle operazioni sui file, tra cui

- la creazione di un file
- la copia di un file
- lo spostamento di un file
- l'eliminazione di file
- l'utilizzo di `FileStream` per leggere e scrivere flussi
  - `StreamReader`
  - `StreamWriter`

La classe `File` è definita nello spazio dei nomi `System.IO`.

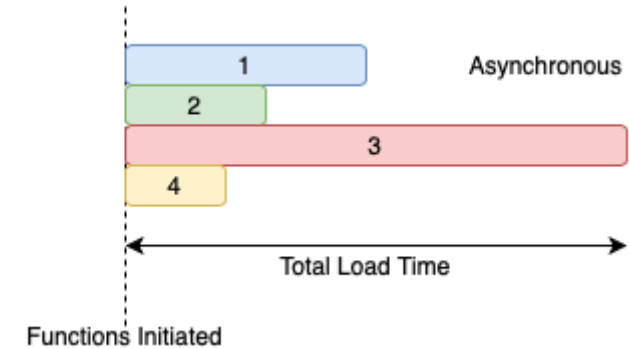
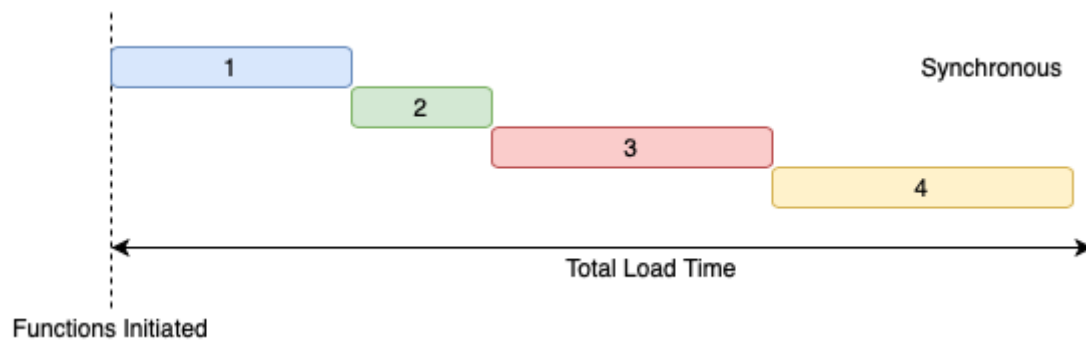


# Accesso ai File

- Se si desidera eseguire operazioni su più file, consultare `Directory.GetFiles` o `DirectoryInfo.GetFiles`
- Il namespace include alcune enumerazioni utilizzate per personalizzare il comportamento di vari metodi di `File`
  - `FileAccess` specifica l'accesso in lettura e/o scrittura a un file
  - `FileShare` specifica il livello di accesso consentito per un file che è già in uso
  - `FileMode` specifica
    - se i contenuti di un file esistente vengono conservati o sovrascritti
    - se le richieste di creazione di un file esistente causano un'eccezione

# Async Await & MultiThreading

- Thread
- Processi
- Esecuzione di codice Asincrono
- Async Await





# Async/await

- Per effettuare e semplificare le chiamate asincrone
- Nuove parole chiave introdotte con .NET Framework 4.5
  - `Async`: gestisce la funzione in asincrono
  - `Await`: attende un'operazione asincrona
- Scriviamo codice come se fosse «sincrono»
- Tutto gestito dal compilatore



# Async/await

- Pattern async-await

```
Public Async Function AccessTheWebAsync() As Task(Of Integer)
    Using client As New HttpClient()

        Dim getStringTask As Task(Of String) =
            client.GetStringAsync("https://docs.microsoft.com/dotnet")

        DoIndependentWork()

        Dim urlContents As String = Await getStringTask

        Return urlContents.Length

    End Using
End Function
```



# Async/await

- Utilizzabile con tutti i metodi \*\*\*Async
  - Restituiscono un riferimento all'operazione, non il risultato
- Normale gestione eccezioni
  - Costrutto try/catch/finally
- Gestione automatica delle problematiche di threading
  - Prima era demandato ad ogni classe (es. WebClient)
  - Per scalare su web e per UI fluide su client



# Codice Parallelo

- Eseguire codice **in maniera parallela**
- **Uscire** dalla sequenzialità
- **Velocizzare** gli accessi alle risorse
- Non bloccare le **UI**
- Eseguire attività “lente” e ottenere i dati **solo al termine** dell'esecuzione
- Sfruttare processori multicore per eseguire un'attività
  - Questo tipo di programmazione accetta un'attività, la suddivide in una serie di più piccole, fornisce istruzioni e i core eseguono le soluzioni contemporaneamente



# Codice Parallelo

- La Task Parallel Library (TPL) è un enorme miglioramento rispetto ai modelli precedenti
- Semplifica l'elaborazione parallela e fa un uso migliore delle risorse di sistema
- Con TPL siamo in grado di implementare la programmazione parallela in C#.NET molto semplice



# Codice Parallelo

## Parallel

- Namespace di riferimento → `System.Threading.Tasks`
- Prevede metodi `Parallel.For` e `Parallel.ForEach` per eseguire cicli
  - `Parallel.ForEach` is for data parallelism
- `Parallel.Invoke` per **invocare differenti metodi** in parallel



# Codice Parallelo

## Parallel.For

```
Public Shared Sub ParallelFor()  
    Dim result As ParallelLoopResult = Parallel.For(  
        0,  
        10,  
        Sub(i)  
            Log($"S {i}")  
            Task.Delay(10).Wait()  
            Log($"E {i}")  
        End Sub)  
    WriteLine($"Is completed: {result.IsCompleted}")  
End Sub
```



# Codice Parallelo

## Parallel.ForEach

```
Public Shared Sub ParallelForEach()  
    Dim data As String() = {"zero", "one", "two", "three",  
"four", "five", "six", "seven", "eight", "nine", "ten",  
"eleven", "twelve"}  
    Dim result As ParallelLoopResult =  
        Parallel.ForEach(data, Sub(s As String)  
                                WriteLine(s)  
                                End Sub)  
End Sub
```



# Codice Parallelo

## Parallel.Invoke

```
Public Shared Sub ParellelInvoke()  
    Parallel.Invoke(New Action(AddressOf Foo), New Action(AddressOf Bar))  
End Sub  
  
Public Shared Sub Foo()  
    WriteLine("foo")  
End Sub  
  
Public Shared Sub Bar()  
    WriteLine("bar")  
End Sub
```

# Codice Parallelo

## Task

- Permettono un maggiore controllo rispetto a Parallel
- Name space di riferimento → System.Threading.Tasks

```
Public Sub TasksUsingThreadPool()  
    Dim tf = New TaskFactory()  
    Dim action As Action(Of String) = AddressOf Foo  
    Dim t1 As Task = tf.StartNew(action, "using a task factory")  
    Dim t2 As Task = Task.Factory.StartNew(action, "factory via a task")  
    Dim t3 = New Task(action, "using a task constructor and Start")  
    t3.Start()  
    Dim t4 As Task = Task.Run(  
        Sub()  
            action("using the Run method")  
        End Sub)  
End Sub
```



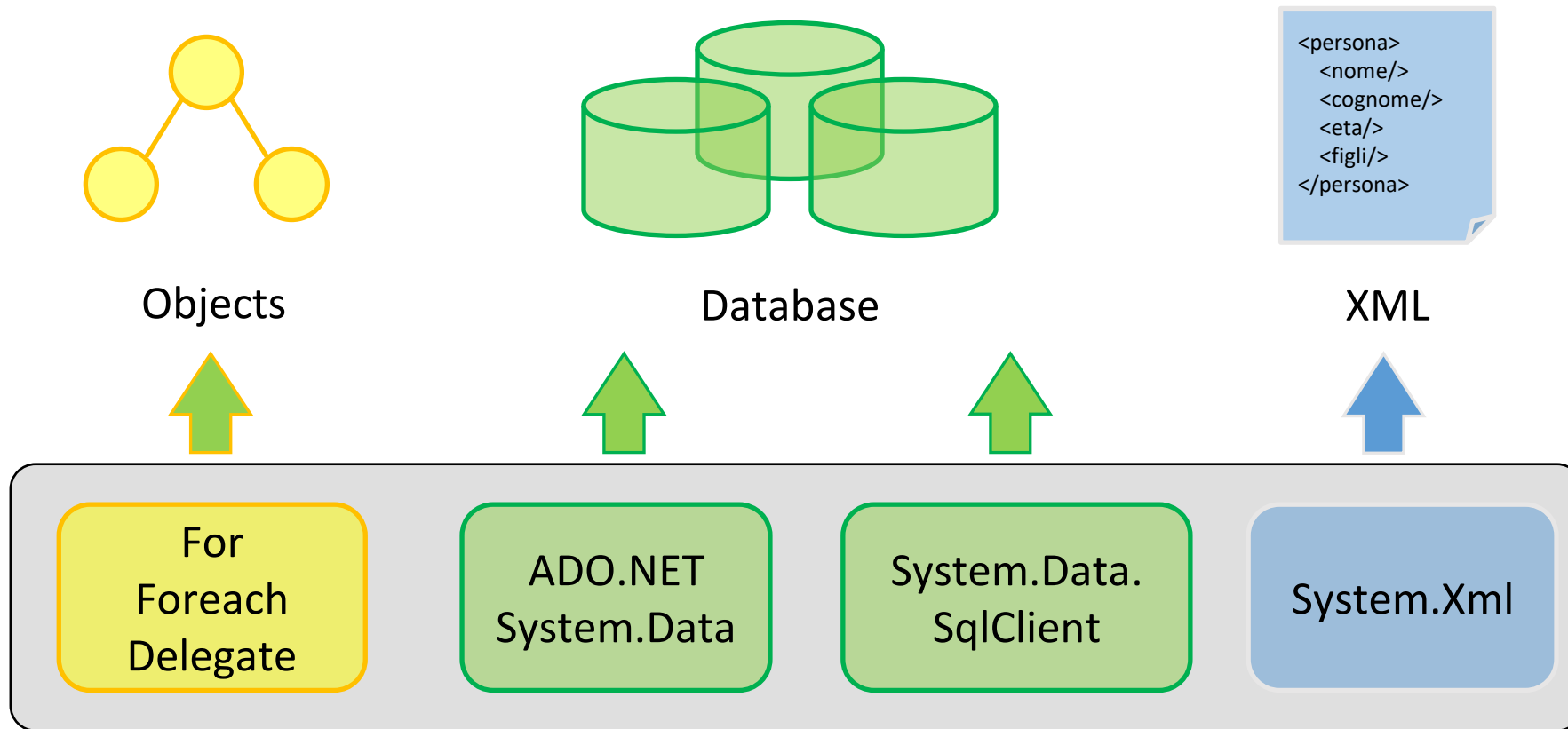
# Cosa è LINQ

LINQ sta per **L**anguage **I**ntegrated **Q**uery

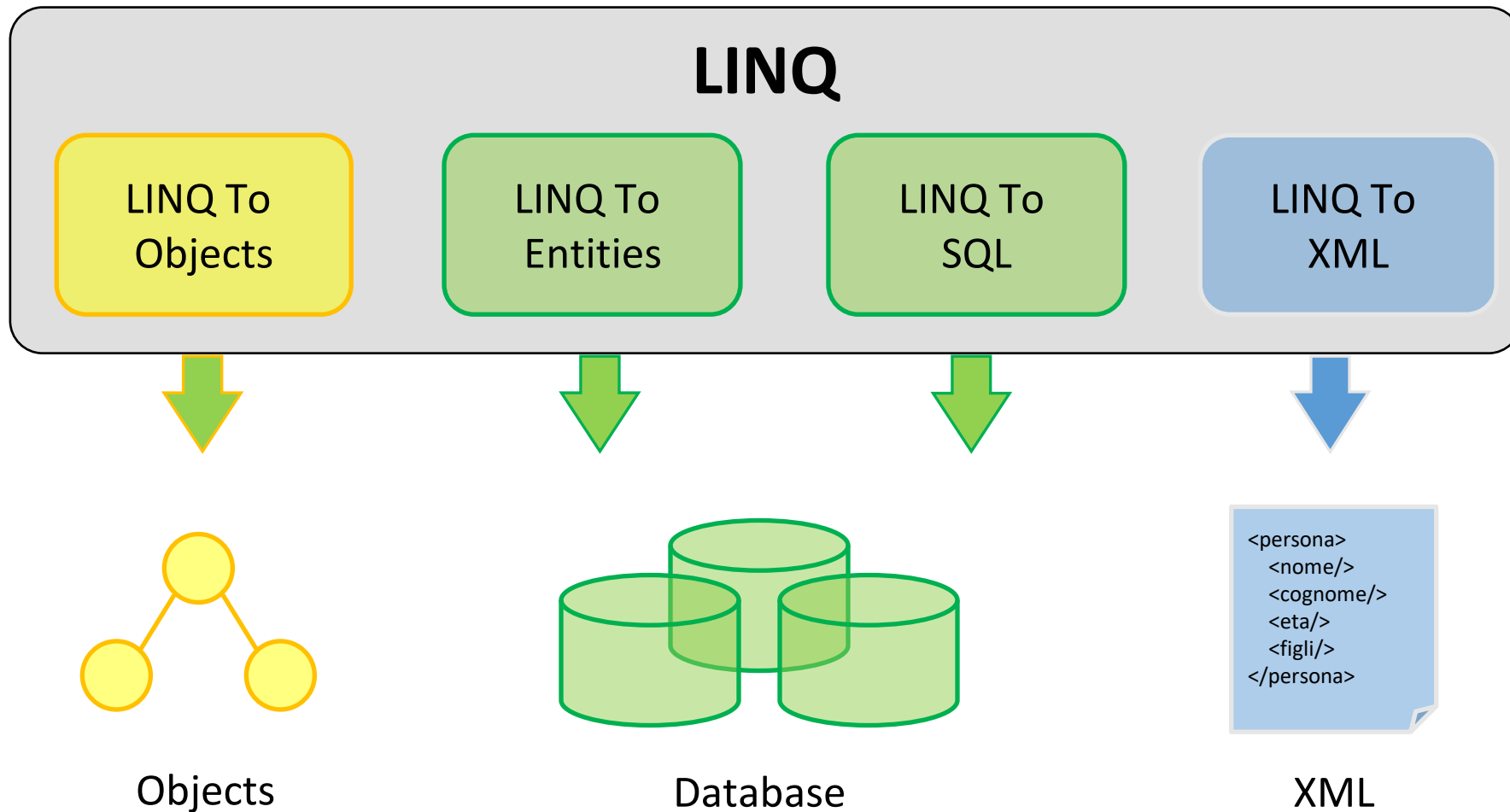
LINQ è un framework per eseguire interrogazioni su sorgenti dati all'interno del linguaggio.



# Accesso ai dati senza LINQ



# Accesso ai dati con LINQ





# LINQ – Query Expression

**Query standard** per accedere a:

- Oggetti
- Dati relazionali
- Dati XML

Più di **50 operatori predefiniti**

- Aggregazione, Proiezione, Join, Partizionamento, Ordinamento

Sintassi e operatori **simile a SQL**

# LINQ – Anatomia di una Query

- Due modelli di sintassi
  - Query
  - Lambda Expression
- Possibilità di utilizzare combinate
- Non modificano la sequenza originale

## Query Lambda

- Più controllo e flessibilità
- Gli operatori sono applicati in sequenza
- **Select** può essere opzionale



# LINQ – Operatori

- Utilizzo di **operatori Standard**
- Libreria di riferimento **System.Linq**
- Utilizzo con tipi **IEnumerable(Of String)**
- Pieno supporto ed integrazione con Intellisense

# Operatori



Tipologia	Operatore
<b>Projection</b>	Select, SelectMany, (From)
<b>Ricerca</b>	Where
<b>Ordinamento</b>	OrderBy, OrderByDescending, Reverse, ThenBy, ThenByDescending
<b>Raggruppamento</b>	GroupBy
<b>Aggregazione</b>	Count, LongCount, Sum, Min, Max, Average, Aggregate,
<b>Paginazione</b>	Take, TakeWhile, Skip, SkipWhile
<b>Insiemistica</b>	Distinct, Union, Intersect, Except
<b>Generazione</b>	Range, Repeat, Empty
<b>Condizionali</b>	Any, All, Contains
<b>Altri</b>	Last, LastOrDefault, ElementAt, ElementOrDefault, First, FirstOrDefault, Single, SingleOrDefault, SequenceEqual, DefaultIfEmpty



# LINQ - Operatori

- Reference: **System.Linq**
- Estende le funzionalità di **IEnumerable<T>** e **IQueryable<T>**

```
Namespace System.Linq
Public NotInheritable Class Enumerable
Public Shared Function Where(Of TSource)(source As IEnumerable(Of TSource), predicate As Func(Of
TSource, Boolean)) As IEnumerable(Of TSource)
```

# Linq - Operatori



```
Namespace System.Linq
... Public NotInheritable Class Enumerable
... Public Shared Function Where(Of TSource)(source As IEnumerable(Of TSource)) As IEnumerable(Of TSource)
... Public Shared Function Min(source As IEnumerable(Of Integer?)) As Integer?
... Public Shared Function Min(source As IEnumerable(Of Long)) As Long
... Public Shared Function Min(source As IEnumerable(Of Long?)) As Long?
... Public Shared Function Min(source As IEnumerable(Of Single)) As Single
... Public Shared Function Min(source As IEnumerable(Of Single?)) As Single?
... Public Shared Function Min(source As IEnumerable(Of Double)) As Double
... Public Shared Function Min(source As IEnumerable(Of Double?)) As Double?
... Public Shared Function Min(source As IEnumerable(Of Decimal)) As Decimal
... Public Shared Function Min(source As IEnumerable(Of Integer)) As Integer
... Public Shared Function Min(source As IEnumerable(Of Decimal?)) As Decimal?
... Public Shared Function Min(Of TSource)(source As IEnumerable(Of TSource)) As TSource
... Public Shared Function Min(Of TSource)(source As IEnumerable(Of TSource)) As TSource
... Public Shared Function Min(Of TSource)(source As IEnumerable(Of TSource)) As TSource
... Public Shared Function Min(Of TSource)(source As IEnumerable(Of TSource)) As TSource
```





# LINQ

Sostituzione di **foreach** con query Linq

## Query Deferred

- Query expression come se dati
- Composizione di query

### Definizione

```
Dim employee As IEnumerable(Of Person) =  
    From p In people  
    Where p.Name = "Scott"  
    Select p.Name
```

### Esecuzione

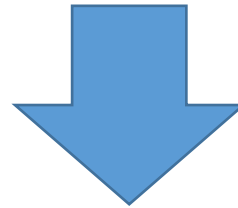
```
For Each emp In employee  
    'code  
Next
```



# LINQ – Lambda Expression

```
Dim filteredList As IEnumerable(Of String) = people.Where(AddressOf StartWithL)

Function StartWithL(p As Person) As Boolean
    Return p.Name.StartsWith("L")
End Function
```



```
Dim filteredList As IEnumerable(Of String) =
    people.Where(Function(p) p.Name.StartsWith("L"))
```



# LINQ – Lambda Expression

- Rappresentazione sintetica
- Utilizzo dell'operatore **Function(x)**
  - **A sinistra:** firma della funzione
  - **A destra:** statement della funzioni



# LINQ – Lambda Expression

## Parametri ed i tipi opzionali

- Non sono richieste parametri, quando sono impliciti

## Logica negli statement

- Utilizzo di variabili locali
- Attenzione: le lambda expression dovrebbero essere tenute più semplici possibile

```
Dim filteredList As IEnumerable(Of String) =  
    people.Where(  
        Function(p)  
            Dim temp As String = p.Name  
            Return temp.StartsWith("L")  
        End Function)
```



# LINQ – Lambda Expression

Lambda Expression usano particolari **delegate**:

- **Action(Of ...)**
  - Non ritornano un valore
- **Func(Of ...) e Expression(Of ...)**
  - Ritornano un valore

```
Dim square As Func(Of Integer, Integer) = Function(x) x * x
Dim mult As Func(Of Integer, Integer, Integer) = Function(x, y) x * y
Dim print As Action(Of Integer) = Sub(x) Console.WriteLine(x)

print(square(mult(3, 5)))
```

# LINQ – Query Expression



- **Extension Methods**
- **Lambda expressions**
  - Delegati
  - Expression Trees
- **Query Expression**

# LINQ – Query Expression



- Extension Methods
- Lambda expressions
- **Query Expression**


```
Dim filterPeople As IEnumerable(Of String) =  
    From person In people  
    Where  
        person.Name.StartsWith("L") AndAlso  
        person.Name.Length < 15  
    Order By person.Name  
    Select person
```



# LINQ – Esecuzione differita

```
Dim allAuthors =  
    From a In authorTable  
    Where a.Id = 1  
    Order By a.Id  
    Select a
```

**allAuthors**: è un'espressione!

```
For Each author In allAuthors  
      
Next
```

Eseguita una query **ogni volta che si accede** alla variabile

```
Dim allAuthors =  
    From a In authorTable  
    Where a.Id = 1  
    Order By a.Id  
    Select a
```

```
allAuthors.ToList()
```



# Domande?

