Introduzione a C# .NET

Presentazione di Fabio Iotti

Argomenti

- Cos'è C#
  - Caratteristiche del linguaggio
  - Esempi di codice in presentazione
- Esempi di codice in Visual Studio
  - Struttura di un progetto .NET (progetto Hello World)
  - Selezione ed iterazione (progetto Guess the Number)
  - Classi (progetto Calculator)
  - Dipendenze e NuGet (progetto Dependencies)
  - Array, liste, set, dizionari (progetto Collections)
  - Funzioni e delegati (progetto Lambda)
  - Generics (progetto Generics)
  - Linq (progetto Linq)

Cos'è C#?

- Linguaggio di programmazione general-purpose
- Sviluppato dalla Microsoft a partire dal 2000
- Strongly-typed
- Orientato ad oggetti (OOP)
- Bytecode (CIL) con virtual-machine (CLR + JIT)
- Garbage-collected
- Open source
- Cross-platform
- Ambiente di sviluppo su Windows: Visual Studio

General-purpose / domain-specific

General-purpose

- Pensato per qualsiasi genere di software
- C/C++
- Python
- Java
- C#

Domain-specific

- Specifico per un determinato dominio
- SQL
- PHP
- MATLAB

Sviluppato dalla Microsoft nel 2000

- Ispirato da Java e C++
- Linguaggio strettamente legato allo sviluppo di .NET Framework

https://it.wikipedia.org/wiki/C_sharp#Storia_del_linguaggio

Esempio di programma C#

```
1  Console.Write("Nome: ");
2  string? name = Console.ReadLine();
3
4  Console.Write("Cognome: ");
5  string? surname = Console.ReadLine();
6
7  Console.WriteLine($"Hello, {name} {surname}!");
```

```
Output:
Nome: Mario
Cognome: Rossi
Hello, Mario Rossi!
```

Strongly-typed

test.py

```python
# Python (weakly-typed)
x = 1
print("x è %s" % (x))

x = "Hello"
print("x è %s" % (x))
```

Output:

```
x è 1
x è Hello
```

Program.cs

```
// C# (strongly-typed)
int x = 1;
Console.WriteLine($"x è {x}");

x = "Hello";
Console.WriteLine($"x è {x}");
```

```
Cannot implicitly convert type 'string' to 'int'
```

Orientato agli oggetti (OOP)

- Classi (e interfacce)
  - Attributi (campi e proprietà)
  - Metodi
- Oggetti
- Ereditarietà
- Polimorfismo

https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti
https://learn.microsoft.com/dotnet/csharp/fundamentals/object-oriented

Classi

Cane.cs

```cs
class Cane {
    public string Nome;

    public Cane(string nome) {
        Nome = nome;
    }

    public void Abbaia() {
        Console.WriteLine("Bau, bau!");
    }
}
```

Program.cs

```cs
Cane mioCane = new Cane("Fuffi");
Console.WriteLine($"Il mio cane si chiama {mioCane.Nome}");
mioCane.Abbaia();
```

Output:

```
Il mio cane si chiama Fuffi
Bau, bau!
```

Attributi (campi / proprietà)

```
1  class Prova {
2      public string Campo;
3      public string Proprietà { get; set; }
4
5      public Prova() {
6          Campo = "Hello";
7          Proprietà = "World";
8      }
9  }
```

- A livello di codice sono simili, ma le proprietà sono più flessibili.
- Dietro le quinte, una proprietà è equivalente ad un campo privato più due metodi (getter e setter).
- È generalmente preferibile usare proprietà piuttosto che campi.

```
1  class Prova {
2      public string Campo;
3      private string proprietàField;
4      public string GetProprietà() {
5          return proprietàField;
6      }
7      public void SetProprietà(string value) {
8          proprietàField = value;
9      }
10
11     public Prova() {
12         Campo = "Hello";
13         SetProprietà("World");
14     }
15 }
```

Metodi

Program.cs

```
1  Cane mioCane = new Cane("Fuffi");
2  Console.WriteLine($"Il mio cane si chiama {mioCane.Nome}");
3  mioCane.Abbaia();
4  mioCane.CambiaNome("Pluto");
5  Console.WriteLine($"Ora il mio cane si chiama {mioCane.Nome}");
```

Cane.cs

```
1  class Cane {
2      public string Nome { get; set; }
3
4      public Cane(string nome) {
5          Nome = nome;
6      }
7
8      public void Abbaia() {
9          Console.WriteLine("Bau, bau!");
10     }
11
12     public void CambiaNome(string nuovoNome) {
13         Nome = nuovoNome;
14     }
15 }
```

Output:

```
Il mio cane si chiama Fuffi
Bau, bau!
Ora il mio cane si chiama Pluto
```

Ereditarietà

Animale.cs

```cs
abstract class Animale {
    public abstract string Specie { get; }

    public abstract void Parla();
}
```

Cane.cs

```cs
class Cane : Animale {
    public override string Specie => "Canide";
    public string Nome { get; set; }

    public Cane(string nome) {
        Nome = nome;
    }

    public override void Parla() {
        Console.WriteLine("Bau, bau!");
    }
}
```

Gatto.cs

```cs
class Gatto : Animale {
    public override string Specie => "Felino";
    public string Nome { get; set; }

    public Gatto(string nome) {
        Nome = nome;
    }

    public override void Parla() {
        Console.WriteLine("Meow!");
    }
}
```

Program.cs

```cs
Animale animale1 = new Cane("Pluto");
Animale animale2 = new Gatto("Romeo");

Console.WriteLine($"Il primo animale è un {animale1.Specie}");
animale1.Parla();

Console.WriteLine($"Il secondo animale è un {animale2.Specie}");
animale2.Parla();
```

Output:

```
Il primo animale è un Canide
Bau, bau!
Il secondo animale è un Felino
Meow!
```

Polimorfismo

Program.cs

```
1  void StampaSpecie(Animale animale) {
2      Console.WriteLine($"La specie è: {animale.Specie}");
3  }
4
5  Cane cane = new Cane("Pluto");
6  Gatto gatto = new Gatto("Romeo");
7
8  StampaSpecie(cane);
9  StampaSpecie(gatto);
```

Output:

```
La specie è: Canide
La specie è: Felino
```

Interfacce

Animale.cs

```
1  abstract class Animale {
2      public abstract string Specie { get; }
3
4      public abstract void Parla();
5  }
```

IAnimale.cs

```
1  interface IAnimale {
2      string Specie { get; }
3
4      void Parla();
5  }
```

- Una sola classe base
- Zero o più interfacce
- La classe base può contenere codice e dati
- Le interfacce non possono contenere codice né dati*

*in realtà da C# 8 le interfacce possono contenere codice, con alcune limitazioni, comunque non possono contenere dati

Bytecode (CIL) eseguito su virtual-machine (CLR + JIT)

- Sorgenti (file con estensione .cs)
- Compilatore (Roslyn / MSBuild)
- Assembly (file con estensione DLL, contengono bytecode)
- Virtual-machine (CLR + JIT)

Garbage-collected

- Memoria gestita implicitamente (niente malloc, free o delete)
- Quando un’area di memoria non è più utilizzata viene liberata
- Garbage collector
- Aree di memoria inutilizzate non rilevate immediatamente
- Il garbage collector parte automaticamente quando necessario

Open Source

- .NET Framework (Microsoft, non open source)
- Mono (originariamente non Microsoft, open source)
- .NET Core (Microsoft, open source)

https://github.com/dotnet

Cross Platform

- .NET Core
- Visual Studio su Windows
- Visual Studio Code su Windows, Mac e Linux
- Runtime compatibile con le principali piattaforme (Windows, Linux, Mac, Android)…
- …ed architetture (x86, x86_64, ARM)
- Xamarin su iOS con AOT

Modificatori di visibilità

- public – visibile ovunque
- protected – visibile all'interno della stessa classe e sottoclassi
- private – visibile solo all'interno della stessa classe
- internal – visibile ovunque nello stesso assembly
- protected internal – visibile ovunque nello stesso assembly, e visibile nelle sottoclassi anche se sono definite in altri assembly

La visibilità di default delle classi è internal, ovvero di default le classi sono visibili solo alle altre classi definite all'interno dello stesso assembly.

Per esporre classi ad assembly esterni, devono essere contrassegnate esplicitamente come public.

All'interno delle classi, tutti i campi, proprietà e metodi sono private.

Per esporli alle sottoclassi devono essere contrassegnati come protected.

Per esporli a tutti devono essere contrassegnati come public.

All'interno delle interfacce tutto è public di default.

E ora...

Spostiamoci su Visual Studio

Risorse esterne

Repository esempi https://github.com/corso-bdx

Microsoft .NET https://dotnet.microsoft.com/

ILSpy https://github.com/icsharpcode/ILSpy

Grazie!

Presentazione di Fabio Iotti