Academy .NET

Ereditarietà e Metodi Virtuali

Sommario

- Ereditarietà
- Polimorfismo
- Superclasse e Classi Derivate
- Metodi Virtuali

Ereditarietà e Polimorfismo

Il linguaggio C# supporta tutti i concetti del paradigma OOP.

Ereditarietà e **Polimorfismo** permettono l'implementazione di classi a partire da un classe esistente, evitando la ripetizione di codice ed ampliandone le funzionalità.

Ereditarietà

Permette di:

Creare una superclasse e quindi definire classi specializzate che hanno accesso ai membri della superclasse

Associata a una relazione "is a"

La classe specializzata "is a" deriva dalla classe generale

Ereditarietà

L'ereditarietà permette la creazione di **gerarchie di tipi**.

- Le **classi derivate**(o sottoclassi) possono ereditare da una singola classe madre
- Ereditano da essa campi e metodi non privati
- Campi e metodi possoo essere personalizzati

Diverse funzionalità comuni alle sottoclassi (o classi figlie) vengono definite e implementate nella superclasse(o classe madre) in quanto ne è una loro **generalizzazione**

Superclasse e Classi Derivate

Classe B **deriva** da Classe A

```
public class A
{

//membri di A

}

class B: A
{

}
```

Classe madre

Classe derivata

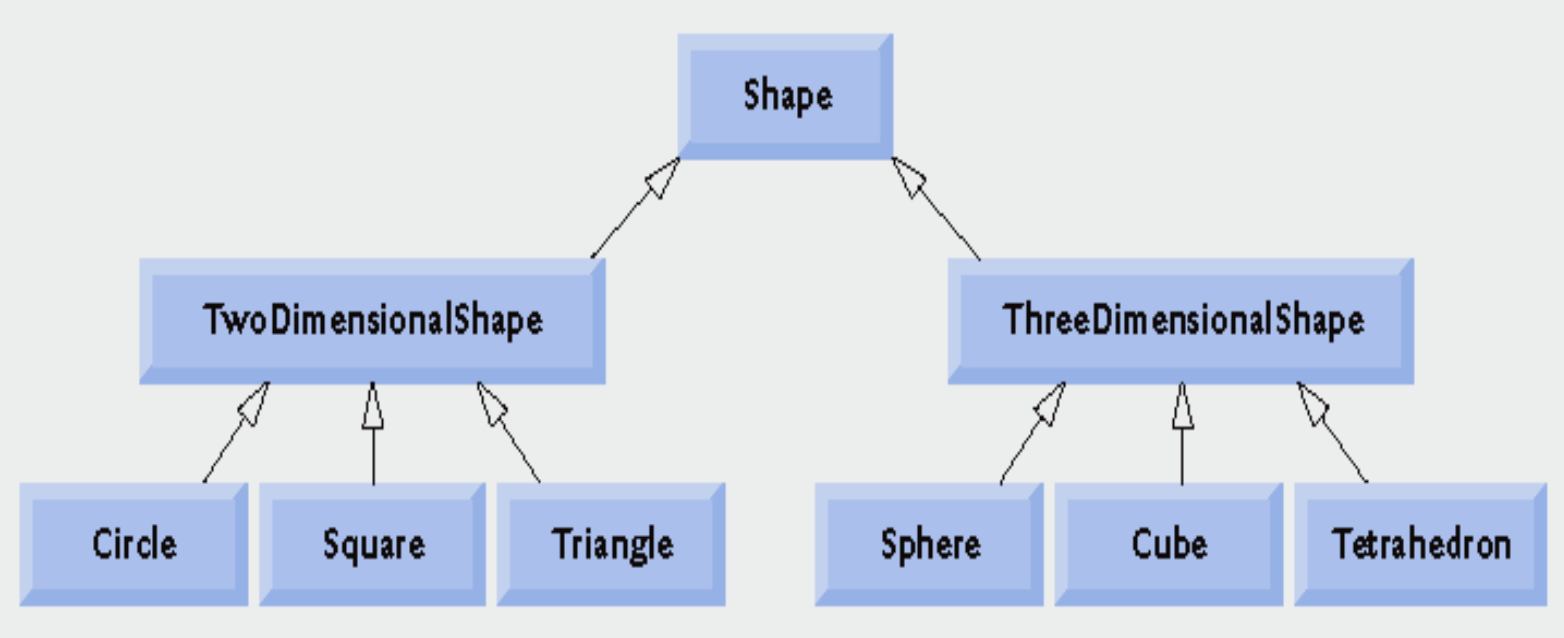
System.Object

- Ogni classe ha come superclasse in C# **System.Object.**
- C# non supporta l'ereditarietà multipla.

- Classe A equivalente alla precedente:

```
public class A: System.Object
{
…
}
```

Ereditarietà: Esempio



Classi Derivate

La classe figlia può implementare i propri **costruttori**, **attributi** e **metodi** oltre a quelli ereditati dalla classe madre.

Istanziare un oggetto di una classe derivata:

Chiama il costruttore sia per la classe base che per la classe derivata

Il costruttore della classe base verrà eseguito per primo

Classi Derivate

- Qualsiasi classe derivata eredita tutti i dati e i metodi della sua classe base, inclusi dati e metodi privati

- **Non è possibile utilizzare o modificare direttamente dati e metodi privati**

Superclasse

```
using System;

public class ParentClass
{
    public ParentClass()
    {
        Console.WriteLine("Parent Constructor.");
    }

    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}
```

Classe Derivata

```
public class ChildClass : ParentClass
{
    public ChildClass()
    {
        Console.WriteLine("Child Constructor.");
    }
}
                //class Program
    public static void Main()
    {
        ChildClass child = new ChildClass();

        child.print();
    }
```

Creazione di classi base per l'ereditarietà

- Può definire le proprie classi da cui possono ereditare altre classi
- La classe base è anche chiamata superclasse o genitore
- I membri dati vengono definiti con un modificatore di accesso **private**
- I costruttori sono definiti con modificatori di accesso **public**
- Le proprietà offrono l'accesso pubblico ai campi dati

Override dei Metodi della Superclasse

La classe derivata contiene **dati** e **metodi** definiti nella classe base

Polimorfismo:

Utilizzando lo stesso metodo o nome di proprietà per indicare diverse implementazioni

- La classe derivata può sovrascrivere e nascondere metodi e dati della classe base

Ereditarietà: Esempio

```
public class Animal
{
    public void Greet()
    {
        Console.WriteLine("Hello, I'm some sort of animal!");
    }
}

public class Dog : Animal
{

}
```

- Questo esempio definisce una classe Animal, con un semplice metodo per generare un saluto.
- Quindi viene definita una classe Dog derivata: la classe Dog eredita dalla classe Animal.

Ereditarietà: Esempio

```
Animal myAnimal = new Animal();

myAnimal.Greet();

Dog myDog = new Dog();
myDog.Greet();  // use parent class method
```

- Anche se non abbiamo definito un metodo Greet() per la classe Dog, sa ancora come salutarci, perché eredita questo metodo dalla classe Animal.
- Può anche definire un metodo Greet() nella classe Dog ... che sovrascrive quello della classe genitore.

Metodi Virtuali

```
public class Animal
{
    public virtual void Greet()
    {
        Console.WriteLine("Hello, I'm some sort of animal!");
    }
}

public class Dog : Animal
{
    public override void Greet()
    {
        Console.WriteLine("Hello, I'm a dog!");
    }
}
```

Metodi Virtuali

Spiegazione esempio precedente:

É stata aggiunta la parola chiave **virtual** al metodo nella classe Animal e sul metodo nella classe Dog è stata utilizzata la parola chiave **override**.

In C # non è consentito eseguire l'override di un membro di una classe a meno che non sia contrassegnato come **virtual**.

Ereditarietà

É comunque possibile accedere al metodo ereditato, anche quando lo si sovrascrive, utilizzando la parola chiave **base**.

```
public override void Greet()
{
    base.Greet();
    Console.WriteLine("Yes I am - a dog!");
}
```

Ereditarietà

L'ereditarietà non è solo da una classe all'altra: è possibile ottenere una gerarchia di classi, che ereditano l'una dall'altra.

Ad esempio, è possibile creare una classe Puppy, che eredita dalla nostra classe Dog, che a sua volta eredita dalla classe Animal.

In C# non è possibile lasciare che una classe erediti da diverse altre classi contemporaneamente.

L'ereditarietà multipla non è supportata in C#.

Accesso ai Metodi

Accesso ai metodi della classe di base da una classe derivata:

Utilizzare la parola chiave **base** per accedere al metodo della classe genitore.

Override dei Metodi

- Sostituzione del metodo definito a un livello superiore

- Parola chiave **override** inclusa nella classe derivata
  - Il metodo nella classe base include la parola chiave **virtual**
  - L'override è differente dall'overloading
  - I metodi **override** hanno esattamente la stessa firma
  - I metodi in **overloading** hanno stesso nome e parametro/i diversi

Polimorfismo

- Esempio di **polimorfismo**:
  Il metodo **ToString()** può avere molti comportamenti diversi
  ToString() utilizza il modificatore **virtual**, il che implica che qualsiasi classe può eseguirne **l'override**

- Le classi derivate ereditano da una classe base
  Chiamate anche sottoclassi o classi figlie

- Modificatore di accesso **protected**
  Accesso solo alle classi che la derivano
  Accesso per modificare i dati e riutilizzo del codice

Domande & approfondimenti

Academy .NET