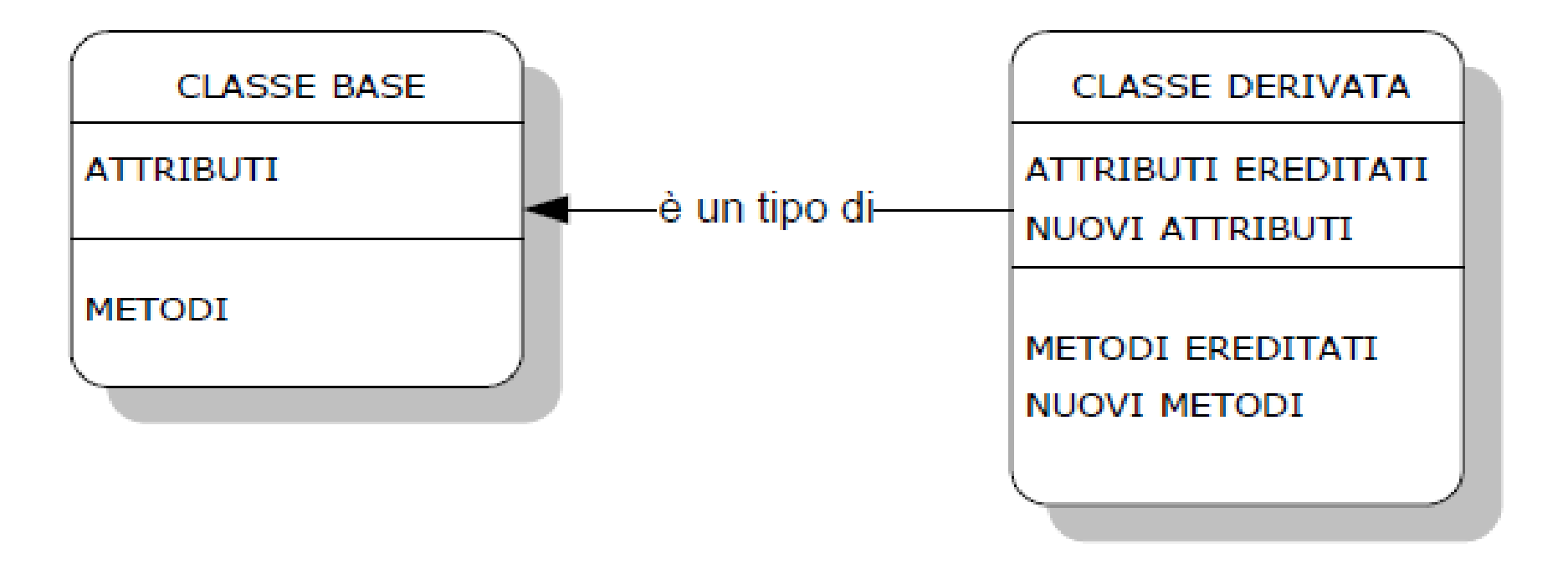
L'ereditarietà

Informatica
Prof. Viglietti Francesco
a.s. 2012-13

Definizione

Si tratta, della possibilità di definire delle classi (derivate o sottoclassi) che discendono da una di livello gerarchicamente superiore (che chiameremo superclasse o base), una specie di relazione padre – figlio, se vogliamo. In questo senso, la classe che discende da una superclasse dovrebbe essere una implementazione, un raffinamento di questa. La classe che eredita dalla base può usufruire di tutti i membri non privati del suo «antenato», potendoli anche modificare oltre ovviamente ad averne di propri. È evidente l'importanza, la comodità e la potenza di questo concetto. Si possono creare lunghe sequenze di discendenti partendo da una classe e via via collegando via ereditarietà una serie di classi di livelli successivi. Ad es. non è difficile intuire la comodità di variare una classe e vedere le modifiche diffuse nelle classi derivate.

Eredita


CLASSE BASE
ATTRIBUTI
METODI
è un tipo di
CLASSE DERIVATA
ATTRIBUTI EREDITATI
NUOVI ATTRIBUTI
METODI EREDITATI
NUOVI METODI

ereditarietà

In C# l'operatore che definisce l'ereditarietà è il **:** (due punti). Concettualmente avremo una situazione di questo tipo, nella sua forma più semplice:

classe A

{

dati pubblici della classe A

}

classe B : A

{

dati della Classe B

}

Cosa significa: abbiamo definito la classe A che avrà la funzione di padre. A ha i suoi metodi pubblici. Poi abbiamo definito la classe B (figlio) la quale oltre che dei suoi dati potrà disporre, in funzione della ereditarietà, anche di quelli della classe A. Banalmente si intuisce che l'ereditarietà si esprime, genericamente come:

class <classe derivata> : <classe base>

Ereditarietà esempio

```
using System;
class A
{
public int x = 1;
public void scrivi()
{
Console.WriteLine("Ciao");
}
}
class B : A
{
public int y = 2;
}
public static void Main()
{
A a = new A();
B b = new B();
Console.WriteLine(a.x);
Console.WriteLine(b.x + " " +
b.y);
b.scrivi();
}
```

L'ereditarietà è realizzata alla riga 10 mentre la 21 e la 22 ci mostrano, che la classe B dispone del membro x e del metodo scrivi. Da questa breve introduzione si può capire quanto siano importanti i modificatori applicati ai campi. Questo discorso, è fondamentale quando si mette in piedi una architettura di un qualche complessità, infatti ogni membro dichiarato pubblico nella classe base sarà pubblico anche nella sua derivata e questo è un fatto di cui tenere conto in quanto, sarà consentito un accesso praticamente indiscriminato. L'esatto opposto avviene se definiamo private i membri della classe padre. Come è noto il modificatore private applicato ai membri rende impossibile la manipolazione di questi al di fuori della classe a cui i membri stessi appartengono. Quindi i derivati ottengono l'eredità si di un membro private ma, al contrario della classe base, non possono accedere ad esso. Anche questo è un aspetto importante da ricordare quando si progetta una classe e si prevede di derivare da essa altre classi. La conseguenza può essere la imprevista indisponbilità di un dato che magari si riteneva di poter usare.

Ereditarietà esempio

```
using System;
class A
{
protected int x = 1;
public void scrivi()
{
Console.WriteLine("Ciao");
}
}
class B : A
{
public int y = 2;
public static void Main()
{
A a = new A();
B b = new B();
Console.WriteLine(b.x + " " + b.y);
b.scrivi();
}
}
```

```
using System;
class Point
{
    protected int x;
    protected int y;
}
class DerivedPoint: Point
{
    static void Main()
    {
        DerivedPoint dp = new DerivedPoint();
        // Direct access to protected members:
        dp.x = 10;
        dp.y = 15;
        Console.WriteLine("x = {0}, y = {1}", dp.x,
dp.y);
    }
}
```

Protected

```
Esempio:
class A
{
private static int x = 1;
public void scrivi()
{
int y = 3 + x;
Console.WriteLine(y);
}
}
class B : A
{
public int y = 2 + x;
}
```

In questo caso non è possibile compilare in quanto la classe B eredita dalla A il membro x che è private in A e quindi non può essere utilizzato nella somma y = 2 + x. Se x fosse public in A allora non ci sarebbero problemi. In questo caso tuttavia x sarebbe accessibile in modo indiscriminato. Se questo non fosse desiderato, un buon compromesso, è ricorrere al modificatore protected che garantisce l'accesso al membro sia da parte della classe base che della sua derivata e da eventuali altre successive.

Va inoltre sottolineato che i costruttori non vengono ereditati; quest'ultimo fatto ha una sua logica, infatti i costruttori servono alla classe cui appartengono è rischioso propagarli. Tuttavia, nelle situazioni in cui risulta utile, è possibile per una sottoclasse richiamare un costruttore della superclasse di appartenenza.