EREDITARIETA’

CORSO DI GAME PROGRAMMING
1° ANNO

Docente **Davide Caio**


Il problema
Ereditarietà
Metodi virtuali
Polimorfismo

IL PROBLEMA

- Stiamo sviluppando un videogioco con diverse entità appartenenti allo stesso gruppo logico, come animali.
- Ogni entità condivide alcune caratteristiche comuni (es. nome, età) e comportamenti simili (es. mangiare, dormire).
- Vogliamo evitare di riscrivere il codice comune in ogni classe specifica (es. Cane, Gatto, Uccello).

- Scrivere codice duplicato aumenta la probabilità di errori e rende le modifiche più difficili.
- Come possiamo gestire le parti comuni in modo efficiente?

EREDITARIETÁ

Definizione: L'ereditarietà è un meccanismo che consente a una classe (detta derivata o figlia) di ereditare attributi e metodi da un'altra classe (detta base o genitore).

Vantaggi:

- Riduce il codice duplicato.
- Facilita la manutenzione del codice.
- Consente di gestire comportamenti comuni a livello della classe base.

Esempio:
Creiamo una classe base Animal che rappresenta le caratteristiche comuni.
Le classi Dog e Cat derivano da Animal e aggiungono caratteristiche specifiche.

EREDITARIETÁ - ESEMPIO

```
public class Animal
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Animal(string name, int age)
    {
        Name = name;
        Age = age;
    }

    public void Eat()
    {
        Console.WriteLine($"{Name} sta mangiando.");
    }

    public void Sleep()
    {
        Console.WriteLine($"{Name} sta dormendo.");
    }
}
```

```
public class Dog : Animal
{
    public Dog(string name, int age) : base(name, age) { }

    public void Bark()
    {
        Console.WriteLine($"{Name} dice: Bau!");
    }
}

public class Cat : Animal
{
    public Cat(string name, int age) : base(name, age) { }

    public void Meow()
    {
        Console.WriteLine($"{Name} dice: Miao!");
    }
}
```

PROTECTED

Permette alle classi derivate di accedere a membri (metodi o attributi) definiti nella classe base ma non accessibili al di fuori di essa.

```
public class Animal
{
    protected string species;

    public Animal(string name, int age, string species)
    {
        Name = name;
        Age = age;
        this.species = species;
    }

    protected void DisplaySpeciesInfo()
    {
        Console.WriteLine($"Specie: {species}");
    }
}

public class Dog : Animal
{
    public Dog(string name, int age) : base(name, age, "Cane")

    public void ShowDetails()
    {
        Console.WriteLine($"Nome: {Name}, Età: {Age}");
        DisplaySpeciesInfo(); // Uso del metodo protetto
    }
}
```

COSTRUTTORE BASE E DEDICATO

Si può definire un costruttore nella classe base e uno nella classe figlia. Il costruttore della classe base sarà poi richiamato durante la costruzione della classe figlia, in questo modo:

```
public Dog(string name, int age) : base(name, age)
{
    // Costruttore dedicato per il cane
}
```

Perché? Perché così è possibile inizializzare gli attributi o lo stato comune ed evitare codice duplicano nella inizializzazione

METODI VIRTUALI E OVERRIDE

Metodi virtuali:

- Definiti nella classe base per consentire l'override nelle classi derivate.
- Forniscono un comportamento predefinito che può essere personalizzato.

```
public class Animal
{
    public string Name { get; set; }

    public Animal(string name)
    {
        Name = name;
    }

    public virtual void Speak()
    {
        Console.WriteLine($"{Name} emette un suono generico.");
    }
}

public class Dog : Animal
{
    public Dog(string name) : base(name) { }

    public override void Speak()
    {
        Console.WriteLine($"{Name} dice: Bau!");
    }
}
```

POLIMORFISMO

Definizione: Il polimorfismo è la capacità di un'istanza di comportarsi come più tipi: un'istanza di una classe derivata può essere trattata sia come il suo tipo specifico che come il tipo della classe base.

Esempio: un oggetto Dog è sia un Dog che un Animal.

```
public class Program
{
    public static void Main()
    {
        Animal myDog = new Dog("Fido");
        Animal myCat = new Cat("Whiskers");

        myDog.Speak(); // Fido dice: Bau!
        myCat.Speak(); // Whiskers dice: Miao!
    }
}
```

METODI VIRTUALI - SPIEGAZIONE

Tabella dei metodi virtuali:

- Ogni classe crea una "vtable" (virtual table) per tracciare i metodi virtuali.
- La vtable associa ogni metodo virtuale a una specifica implementazione.

Esempio di funzionamento:

Durante l'istanziazione di un oggetto (es. Dog), la vtable dell'oggetto viene inizializzata con l'implementazione di Speak specifica di Dog.

Quando viene chiamato il metodo virtuale (Speak) su un riferimento alla classe base (Animal), la vtable dell'oggetto determina quale implementazione chiamare.

ATTENZIONE A COSA POTETE CHIAMARE

Tabella dei metodi virtuali:

- Ogni classe crea una "vtable" (virtual table) per tracciare i metodi virtuali.
- La vtable associa ogni metodo virtuale a una specifica implementazione.

Esempio di funzionamento:

Durante l'istanziazione di un oggetto (es. Dog), la vtable dell'oggetto viene inizializzata con l'implementazione di Speak specifica di Dog.

Quando viene chiamato il metodo virtuale (Speak) su un riferimento alla classe base (Animal), la vtable dell'oggetto determina quale implementazione chiamare.

COME USARE METODI SPECIFICI DI CLASSI DERIVATE

Regola generale:

- Se dichiari una variabile di tipo della classe base e assegni un'istanza di una classe derivata, puoi usare solo i metodi e le proprietà della classe base.

Metodi virtuali e override:

- Se un metodo è virtuale e viene sovrascritto nella classe derivata, verrà utilizzata la definizione della classe derivata.

Limiti:

- Non puoi chiamare metodi specifici della classe derivata senza fare un cast.

COME USARE METODI SPECIFICI DI CLASSI DERIVATE

```
Animal myAnimal = new Dog("Fido");
myAnimal.Speak(); // Output: Fido dice: Bau!
// myAnimal.Bark(); // Errore: il metodo Bark non è definito in

Dog myDog = (Dog)myAnimal; // Cast esplicito
myDog.Bark(); // Output: Fido dice: Bau!
```

- Il cast è necessario per accedere ai metodi o attributi specifici della classe derivata.
- Attenzione: Assicurati che il tipo effettivo dell'oggetto sia compatibile prima di eseguire il cast, altrimenti otterrai un'eccezione a runtime.