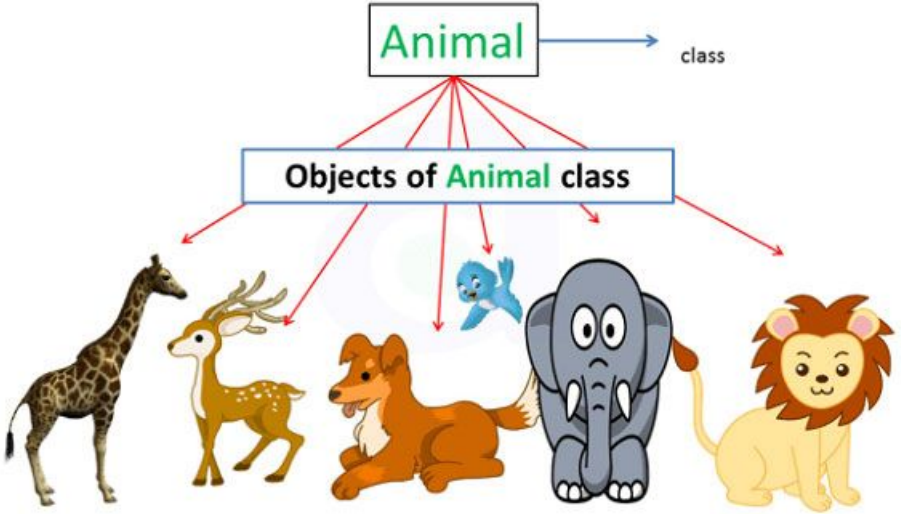
OOP

Ereditarietà
Parte 2

Ereditarietà

Nel OOP si possono creare entità derivanti da altre entità più astratte di livello superiore.

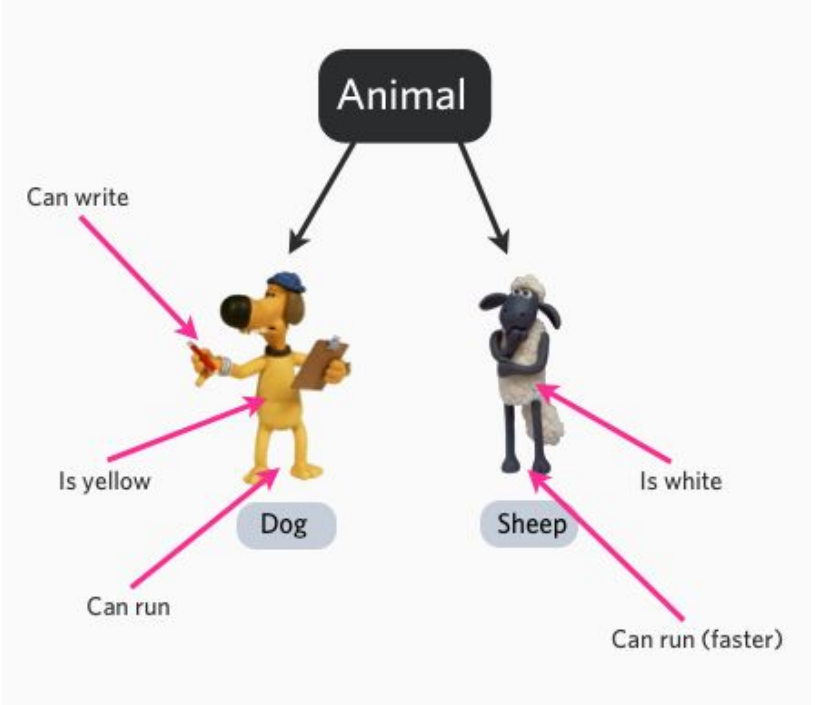
Il **Cane** è un'entità concreta e dettagliata, "figlia" dell'entità più astratta **Mammifero**, la quale si può ricondurre all'entità ancora più astratta **Animale**.



Ereditarietà

Ogni classe che rientra in una catena di ereditarietà **acquisisce** dalla classe "padre", o superiore, quei metodi e attributi marcati come **public** o **protected**.

Ogni classe "figlia" (o concreta) può anche definire nuovi metodi e nuovi attributi che appartengono solamente a lei.



Esempio di ereditarietà

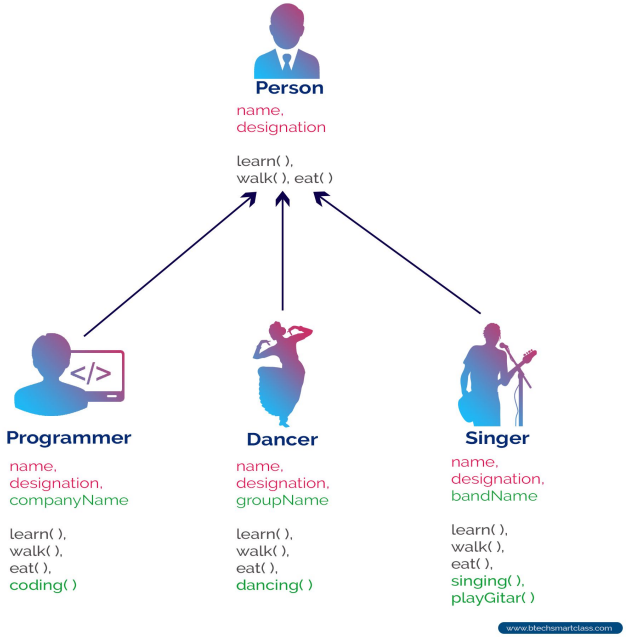
```
public abstract class Person
{
    public string Name { get; private set; }
    public string Designation { get; private set; }

    protected Person(string name, string designation)
    {
        Name = name;
        Designation = designation;
    }

    public string Learn()
    {
        return "I'm learning.";
    }

    public string Walk()
    {
        return "I'm walking.";
    }

    public string Eat()
    {
        return "I'm eating.";
    }
}
```



La classe base **`Person`** rappresenta il modello software di una persona, quindi troppo generico per poter generare oggetti utili ai fini del software. Per questo motivo viene marcata con la direttiva **`abstract`**. In questo modo si impedisce che sia possibile istanziare oggetti di tipo Person.

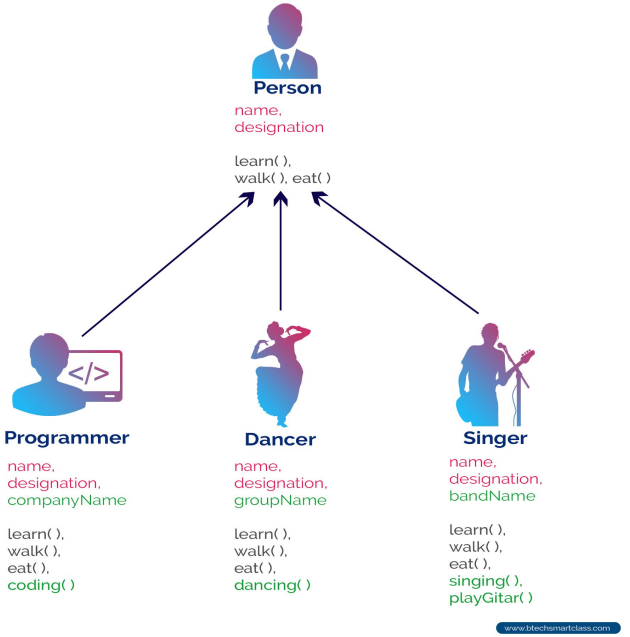
Esempio di ereditarietà

```
public class Programmer : Person
{
    public string CompanyName { get; private set; }
    public Programmer(string name, string role, string companyName) : base(name, role)
    {
        CompanyName = companyName;
    }

    public string Coding()
    {
        return "I'm coding!";
    }
}

public class Dancer : Person
{
    public string GroupName { get; private set; }
    public Dancer(string name, string role, string groupName) : base(name, role)
    {
        GroupName = groupName;
    }

    public string Dancing()
    {
        return "I'm dancing!";
    }
}
```



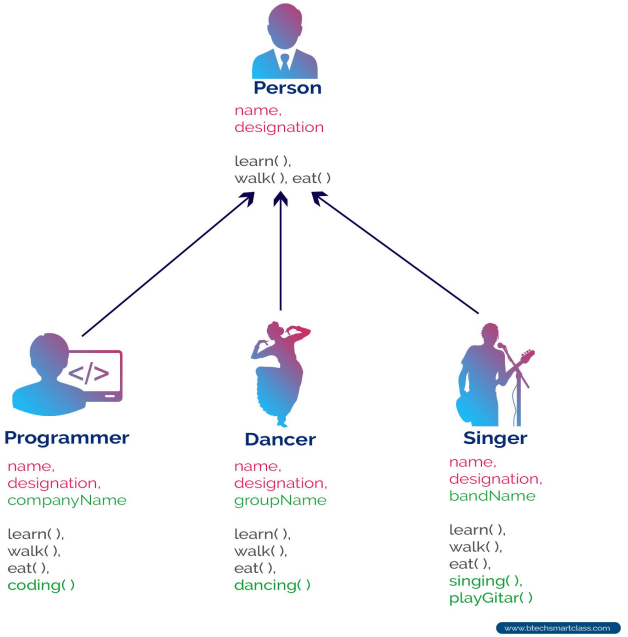
I costruttori delle classi derivate richiamano il costruttore della classe base passando le informazioni name e role, poi la classe derivata si occupa di incapsulare i propri dati aggiuntivi.

Esempio di ereditarietà

```
Programmer prg1 = new Programmer("Annalisa", "Front-end developer", "Programmino S.r.l.");
Console.WriteLine("{0} is a {1} and work in {2}.",
                  prg1.Name, prg1.Designation, prg1.CompanyName);
Console.WriteLine(prg1.Eat());
Console.WriteLine(prg1.Coding());

Dancer dancer1 = new Dancer("Luca", "Hip hop dancer", "Arte e musica");
Console.WriteLine("{0} is a {1} and work in {2}.",
                  dancer1.Name, dancer1.Designation, dancer1.GroupName);
Console.WriteLine(dancer1.Eat());
Console.WriteLine(dancer1.Dancing());
```

```
Annalisa is a Front-end developer and work in Programmino S.r.l..
I'm eating.
I'm coding!
Luca is a Hip hop dancer and work in Arte e musica.
I'm eating.
I'm dancing!
```



I costruttori delle classi derivate richiamano il costruttore della classe base passando le informazioni name e role, poi la classe derivata si occupa di incapsulare i propri dati aggiuntivi.

Ereditarietà e overriding

Esistono casi in cui la **logica** di funzionamento della singola classe concreta non è in comune a tutta la gerarchia ma è **diversa** per ognuna.

Immaginiamo di dover sviluppare delle classi che generano un numero con logiche diverse:

- **RandomNumberGenerator**: genera numeri casuali come fa l'oggetto Random del framework .Net
- **TimedNumberGenerator**: genera numeri casuali basandosi sul tempo
- **FixedNumberGenerator**: genera sempre lo stesso numero

Ereditarietà e overriding

```
public abstract class NumberGenerator
{
    /// <summary>
    /// Seme che inizializza la logica di generazione dei numeri.
    /// </summary>
    protected int Seed { get; }

    protected NumberGenerator(int seed)
    {
        Seed = seed;
    }

    /// <summary>
    /// Metodo che conterrà nelle classi concrete la logica di generazione del numero.
    /// </summary>
    /// <returns>Il numero generato.</returns>
    public abstract int Generate();
}
```

La classe base **NumberGenerator** è astratta, quindi non istanziabile direttamente.
Anche il metodo **Generate** è astratto, quindi le classi derivate saranno obbligate a implementarne la logica.

Ereditarietà e overriding

```
public class FixedNumberGenerator : NumberGenerator
{
    public FixedNumberGenerator(int fixedNumber) : base(fixedNumber) { }

    public override int Generate()
    {
        return Seed;
    }
}
```

Ereditarietà e overriding

```
public class RandomNumberGenerator : NumberGenerator
{
    private Random rnd;

    public RandomNumberGenerator(int seed) : base(seed)
    {
        rnd = new Random(Seed);
    }

    public override int Generate()
    {
        return rnd.Next();
    }
}
```

Ereditarietà e overriding

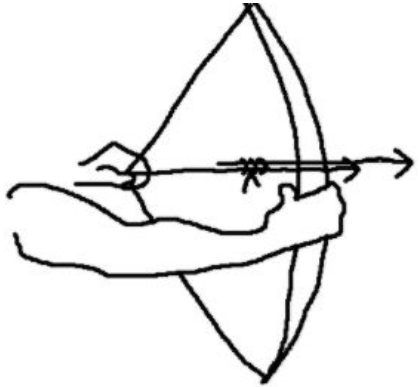
```
public class TimedNumberGenerator : NumberGenerator
{
    public TimedNumberGenerator(int seed) : base(seed) { }

    public override int Generate()
    {
        return (int)DateTime.Now.Ticks / Seed;
    }
}
```

Ereditarietà e overriding

La tecnica dell'overriding si usa anche in quelle situazioni in cui la classe base offre una **logica** di default ma la classe derivata vuole **modificarla** o **ridefinirla** totalmente.

Come esempio prendiamo il caso della classe Person e la classe derivata Programmer.



Overriding

Ereditarietà e overriding

```
public abstract class Person
{
    public string Name { get; private set; }
    public string Designation { get; private set; }

    protected Person(string name, string designation)
    {
        Name = name;
        Designation = designation;
    }

    public virtual string Learn()
    {
        return "I'm learning.";
    }

    public string Walk()
    {
        return "I'm walking.";
    }

    public string Eat()
    {
        return "I'm eating.";
    }
}
```

La classe base deva "autorizzare" l'override del suo metodo con la direttiva **virtual**.

Ogni metodo della classe base che possiede virtual è abilitato ad essere sovrascritto dalle classi derivate, a loro discrezione.

Ereditarietà e overriding

```
public class Programmer : Person
{
    public string CompanyName { get; private set; }
    public Programmer(string name, string role, string companyName)
        : base(name, role)
    {
        CompanyName = companyName;
    }

    public string Coding()
    {
        return "I'm coding!";
    }

    public override void Learn()
    {
        string baseMessage = base.Learn();
        return baseMessage + " I'm a FITSTIC student.";
    }
}
```

La classe derivata fa **override** del metodo della classe base, cambiato l'implementazione di default.

La classe derivata può decidere di richiamare l'implementazione di default oppure ignorarla e ridefinire il metodo con la propria logica.

Esercizi

1. Creare la classe **Punto2D** caratterizzata dagli attributi X e Y quali coordinate cartesiane del punto. La classe deve possedere il metodo double CalcolaDistanza(Punto2D p) che calcola la distanza tra lo stesso punto e il punto p in input.
2. Impostare una gerarchia di classi per rappresentare le figure geometriche triangolo, quadrato, rettangolo, cerchio. Ogni figura si crea fornendo la lunghezza dei lati. Per ogni tipologia di figura deve essere possibile far calcolare perimetro e area.
   Per i triangoli si vuole anche sapere la tipologia (isoscele, equilatero, scaleno) e se è rettangolo oppure no.
   Per i quadrati e i rettangoli si vuole calcolare la diagonale.
3. I quadrati e i rettangoli possono essere visti come la congiunzione di due triangoli rettangoli lungo le rispettive ipotenuse. Implementare sui quadrati e rettangoli il metodo Riduci() che restituisce tale triangolo.
4. Per il triangolo implementare il metodo Componi() che restituisce la figura risultante dal raddoppio del triangolo stesso.
5. Per i quadrati e rettangoli creare l'overload del costruttore che compone la figura a partire da un triangolo rettangolo.
6. Per i triangoli implementare l'overload del costruttore che compone la figura a partire da un quadrato o da un rettangolo.