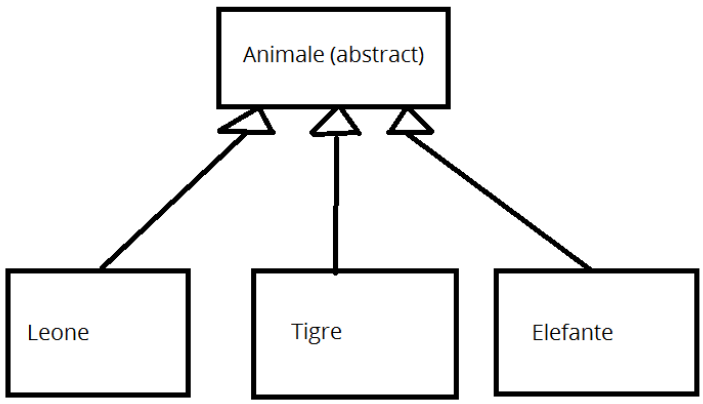
Creare un Oggetto scegliendo da un menu il nome della Classe

Si abbia una gerarchia di classi



La classe base della gerarchia(Animale) è **astratta** in quanto lascia da specificare i dettagli del comportamento dei metodi a cura delle sottoclassi concrete (Leone, Tigre, Elefante).

```
using System;

namespace mio{

public abstract class Animale
{
        // properties
        public string Nome {get; set;}

        // costruttori che servono solo per inizializzare le properties
        // visto che non si possono creare oggetti da una classe astratta
        public Animale()
        { nome = “anonimo”; }
        public Animale(string unNome)
        { nome = unNome; }
        public abstract string FaiVerso();
}

public class Leone: Animale
{
        public int Peso {get; set;}
        public Leone():base()
        { peso = 0; }
        public Leone(int unPeso, string unNome):base(unNome)
        { peso = unPeso;}
        public override string FaiVerso()
        { return “ROARR”; }
}
```

analogamente per gli altri tipi di animali, che avranno eventualmente degli attributi specifici propri, oltre a quello ereditato da Animale (ovvero oltre al Nome).

Si potrebbe creare una classe Zoo per gestire una collezione di animali di diverse specie, con in particolare i metodi AggiungiAnimale(Animale a) per aggiungere un animale alla collezione e FaiConcerto() per far cantare tutti gli animali della collezione.

```
public class Zoo
{
        public List<Animale> Elenco {get; set; }

        public Zoo()
        { Elenco = new List<Animale>(); }

        public void AggiungiAnimale(Animale a)
        {  Elenco.Add(a); }

        public string FaiConcerto()
        {
        string risultato = "";
        Elenco.ForEach( a=> {risultato = risultato + a.FaiVerso();} );
        return risultato;
        }
}
```

Se si vuole scrivere un programma con interfaccia grafica per lavorare con gli animali, dopo aver creato un oggetto Zoo, si dovranno aggiungere degli animali all'Elenco.

```
class Form
{
   private Zoo mioZoo;
   public Form()
   {  mioZoo = new Zoo(); }
}
```

Se si ha un menu a tendina con gli animali da scegliere è facile associare alle diverse voci del menu la corretta chiamata al metodo AggiungiAnimale(), come ad esempio:

```
private void MenuItemLeone_Click(object sender,
RoutedEventArgs e)
{
      mioZoo.AggiungiAnimale(new Leone());
}
```

Se invece si ha un TextBox dove poter digitare il tipo di animale da aggiungere, oppure una ListBox da cui scegliere, si dovrà fare un controllo su che cosa ha scelto l'utente:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
   string s  = listBox1.SelectedItem.ToString();  // ad esempio "Leone"
   if (s == "Leone")
      {  mioZoo.AggiungiAnimale(new Leone()); }
   else if (s == "Tigre")
      {  mioZoo.AggiungiAnimale(new Tigre()); }
   else if (s == "Elefante")
      {  mioZoo.AggiungiAnimale(new Elefante()); }
   else …
}
```

Questo modo di programmare diventa scomodo se, in particolare, si hanno tanti diversi tipi di animali.

Una tecnica più conveniente è basata sulla REFLECTION, ovvero sulla possibilità per un programma di consultare i propri metadati, ovvero la descrizione delle classi che lo compongono.

Pertanto, grazie alla classe System.Type, che rapprenta il tipo di un oggetto e alla classe System.Activator si potrà creare una istanza di una classe di cui si specifica il nome come stringa:

```
// il namespace è "mio" e la classe si chiama "Leone"

Type tipo = Type.GetType("mio.Leone");

Animale a = (Animale) Activator.CreateInstance(tipo);
```

Viene creato un oggetto di tipo Leone, che è assegnato ad una variabile di tipo Animale.

Nel caso in cui il costruttore da utilizzare abbia un paio di parametri, come il secondo costruttore di Leone, che vuole un peso intero e un nome stringa, le istruzioni sarebbero le seguenti:

```
Type tipo = Type.GetType("mio.Leone");
int x = 100;
string y = "elsa";
Animale a = (Animale) Activator.CreateInstance(tipo, x, y);
```

Se il metodo Type.GetType() fallisce ritorna un valore null:

```
Type tipo = Type.GetType("mio.AnimaleInesistente");   // si ottiene null
```

In definitiva, si può scrivere in modo molto più succinto e conveniente il metodo associato alla scelta dalla ListBox:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
   string s  = listBox1.SelectedItem.ToString();  // ad esempio “Leone”
   Type tipo = Type.GetType("mio."+s);
   Animale a = (Animale) Activator.CreateInstance(tipo);
}
```

Questo vale per tutti gli animali presenti nella ListBox e non ci sono problemi se questo elenco varia nel tempo!