Programmazione C#

Lezione 3

Programmare ad Oggetti

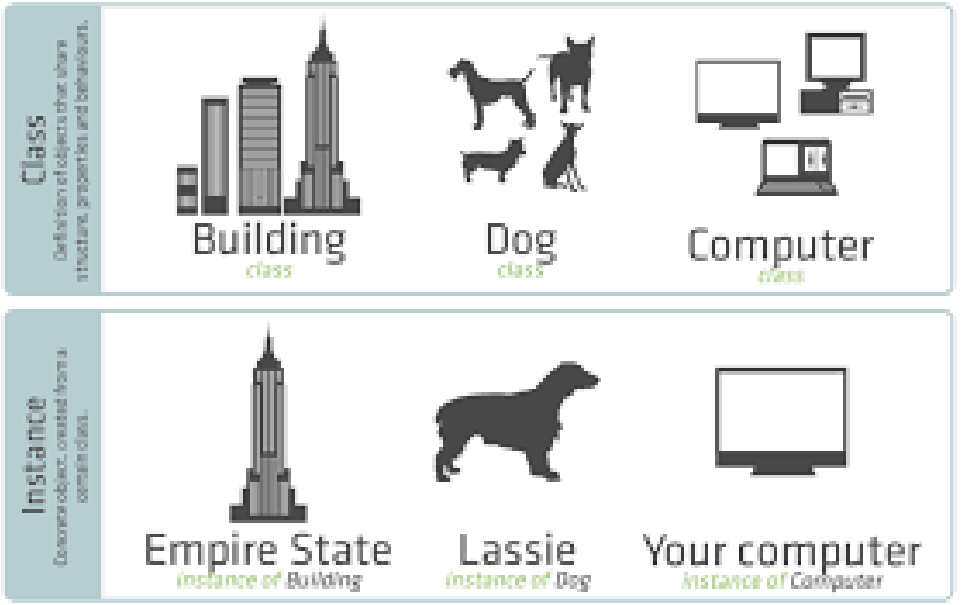
e-Linking online Systems

Sommario della Lezione

1. Introduzione alla Programmazione ad Oggetti
2. Interfacce
3. Strutture

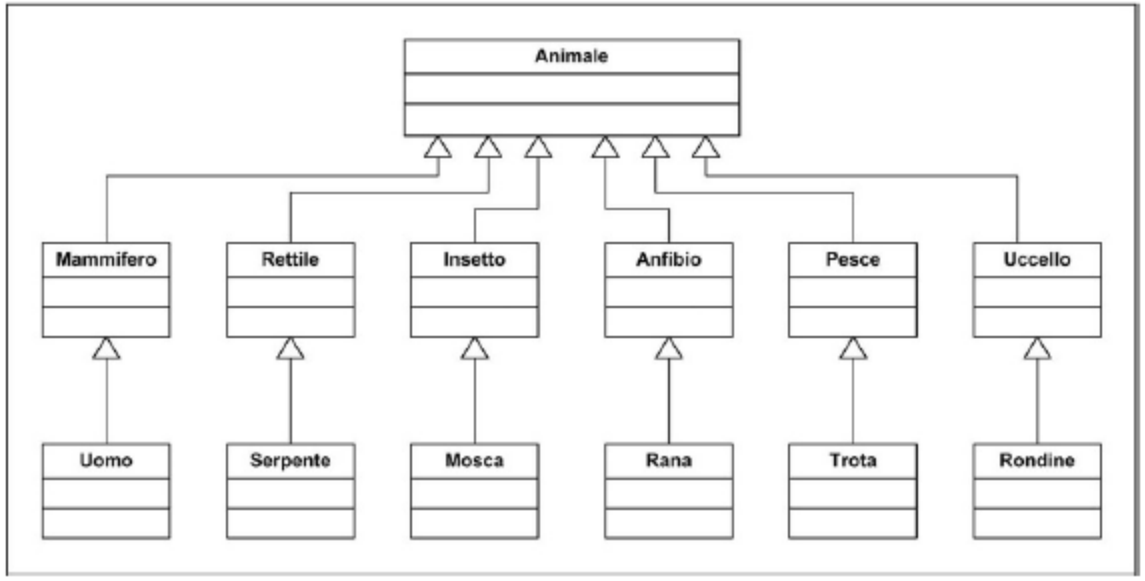
Vantaggi Programmazione ad Oggetti

- La logica utilizzata è simile a quella che «percepiamo»
- Migliore Struttura del Codice
- Possibilità di creare i nostri «tipi»
- Tutti – o quasi – oggi la utilizzano



Principi Fondamentali

- ***Ereditarietà***: E' possibile creare delle classi padri e figlie al fine di far ereditare dei comportamenti/caratteristiche
- ***Polimorfismo***: Principio per cui le classi figlie possono implementare uno stesso comportamento, ereditato dal padre, in modo differente
- ***Incapsulamento***: Principio base per cui una classe può «mostrare» solo quello che effettivamente serve essere mostrato



Classi in C#

```
class Person
{
  // …
}
```

```
public class Person
{
  public string FirstName { get; set; } = "Daniele";
  public string LastName { get; set; } = "Bochicchio";
  public int Age { get; set; } = 36;
  public bool IsActive { get; } = true;
  public string LastName => $"{FirstName} {LastName}";
}
```

Parola chiave	Visibilità
`public`	Da tutte le classi
`protected`	Solo dalle classi derivate
`private`	Non accessibile
`internal`	All'interno dell'assembly
`protected internal`	Combinazione delle due

Costrutture di una Classe e Inizializzazione

```
public class Person
{
  private string _fullName;
  private readonly int _age;
  // Costruttore di default
  public Person()
  {
    _fullName = "Sconosciuto";
    _age = 18;
  }
  // Costruttore con parametri
  public Person(string name, int age)
  {
    _fullName = name;
    _age = age;
  }
}
Person x = new Person();
Person y = new Person("Daniele Bochicchio", 36);
```

```
var persons = new Person[]
{
  new Person { FullName = "Cristian Civera", Age = 34 },
  new Person { FullName = "Daniele Bochicchio", Age = 36 }
};
```

Classi Statiche

```
public class Person
{
  public string FullName { get; set; }
  public Person(string name)
  {
    this.FullName = name;
  }
  // Metodo d'istanza
  public string GetFirstName()
  {
    return this.FullName.Split(' ')[0];
  }
  // Metodo statico
  public static string GetFirstName(string name)
  {
    return name.Split(' ')[0];
  }
}
```

```
public static class Calculator
{
  public static int Sum(int x, int y)
  {
    // Ritorna la somma dei due numeri
    return x + y;
  }
  public static int Subtract(int x, int y)
  {
    // Ritorna la differenza tra i due numeri
    return x - y;
  }
}
int i = Calculator.Sum(18, 8);
int j = Calculator.Subtract(18, 8);
```

Classi Statiche

```
public class Person
{
  public string FullName { get; set; }
  public Person(string name)
  {
    this.FullName = name;
  }
  // Metodo d'istanza
  public string GetFirstName()
  {
    return this.FullName.Split(' ')[0];
  }
  // Metodo statico
  public static string GetFirstName(string name)
  {
    return name.Split(' ')[0];
  }
}
```

```
public static class Calculator
{
  public static int Sum(int x, int y)
  {
    // Ritorna la somma dei due numeri
    return x + y;
  }
  public static int Subtract(int x, int y)
  {
    // Ritorna la differenza tra i due numeri
    return x - y;
  }
}
int i = Calculator.Sum(18, 8);
int j = Calculator.Subtract(18, 8);
```

Classi Parziali

```
public partial class Person
{
  public void Write()
  {
    // …
    OnWrite();
    // …
  }
  // Dichiarazione del metodo parziale privato
  partial void OnWrite();
}
public partial class Person
{
  partial void OnWrite()
  {
    // Implementazione
  }
}
```

Classi Parziali

```
public partial class Person
{
  public void Write()
  {
    // …
    OnWrite();
    // …
  }
  // Dichiarazione del metodo parziale privato
  partial void OnWrite();
}
public partial class Person
{
  partial void OnWrite()
  {
    // Implementazione
  }
}
```

Ereditarietà

```
public class Employee : Person
{
  private string _firstName;
  public Employee(string name, int age)
  {
    base.FullName = name;
    base.Age = age;
    _firstName = base.GetFirstName();
  }
}
```

```
// Person è una classe astratta e deve essere derivata
public abstract class Person
{
  // Metodo astratto
  public abstract GetFirstName();
  // Proprietà astratta
  public abstract string FullName { get; set; }
}
```

```
// Customer (cliente) deriva da Person e non può avere classi derivate
public sealed class Customer : Person
{
    // …
}
```

```
public class Person
{
  // Metodo virtuale
  public virtual string GetFirstName()
  {
    // Implementazione di base del metodo
  }
}
public class Employee : Person
{
  public Employee(string name, int age)
  {
    // …
  }
  public override string GetFirstName()
  {
    // Nuova implementazione del metodo
  }
}
Employee x = new Employee("Riccardo Golia", 41);
Person y = x;
// Viene comunque eseguito il metodo di Employee
// anche se il riferimento è di tipo Person
string z = y.GetFirstName();
```

Interfacce

```
// Interfaccia che definisce un metodo per la scrittura
public interface IWritable
{
  void Write();
}
```

```
// Deriva dalla classe base Person
// Implementa l'interfaccia IWritable
public class Employee : Person, IWritable
{
  public void Write()
  {
    Console.Write(base.FullName);
  }
}
// Il riferimento è di tipo IWritable, ma l'istanza è di tipo Employee
IWritable x = new Employee("Riccardo Golia", 41);
// Scrive il nome completo
x.Write();
// È necessario effettuare il casting per invocare il metodo
string y = ((Employee)x).GetFirstName();
```

Strutture

Classi	Strutture
Possono definire campi, proprietà, metodi ed eventi.	Possono definire campi, proprietà, metodi ed eventi.
Supportano tutti i tipi di costruttori e l'inizializzazione dei membri.	Supportano tutti i tipi di costruttori (novità di C# 6) e l'inizializzazione dei membri.
Supportano il metodo `Finalize`, ovvero il distruttore.	Non supportano il metodo `Finalize`.
Supportano l'ereditarietà.	Non supportano l'ereditarietà.
Sono tipi di riferimento.	Sono tipi di valore.
Vengono allocati nel managed heap.	Vengono allocati sullo stack.

Strutture

```
// Definizione di una struttura per i numeri complessi
public struct Complex
{
  public float Real { get; set;} // Parte reale
  public float Imaginary { get; set;} // Parte immaginaria
  // Costruttore con parametri
  // Deve essere richiamato esplicitamente il costruttore di
  // default per poter utilizzare le proprietà automatiche
  public Complex(float r, float i) : this()
  {
    this.Real = r;
    this.Imaginary = i;
  }
 // Metodo statico per la somma di due numeri complessi
 public static Complex Sum(Complex x, Complex y)
 {
   return new Complex(x.Real + y.Real, x.Imaginary + y.Imaginary);
 }
}
Complex u = new Complex();   // u vale 0.0 + 0.0i
Complex v = new Complex(1.0F, 2.0F); // v vale 1.0 + 2.0i
```

Fine Lezione 3

DOMANDE?