OOP

LEZIONE 2

Focus on…

- Classe è un tipo di dato oggetto
- Istanza è la variabile di tipo oggetto
- Ogni classe ha un costruttore; se non esplicito il sistema ne associa uno in automatico DEFAULT
- Il costruttore serve per allocare memoria per le istanze
- Il costruttore è omonimo della classe
- È possibile l'overload dei costruttori
- Le parti dell'oggetto possono essere private o pubbliche; se non si specifica nessuna opzione si presuppone siano private
- Il meccanismo di parti private è detto incapsulamento

Metodi d’istanza

- Le operazioni che un oggetto può svolgere hanno il nome di **metodi**. In sintesi un metodo è un’operazione che, a partire da oggetto, esegue elaborazione e può restituire un risultato e/o un effetto sui dati dell’oggetto.

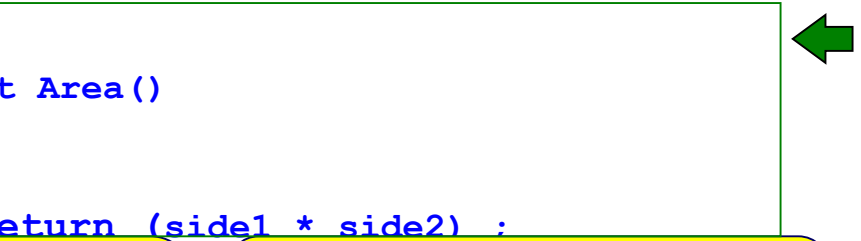
VC#

```
public class Rectangle
{
   private double side1, side2;
   public double Area ( )
   {
      //corpo del metodo
      return side1 * side2 ;
   }
}
```

- Nell’esempio precedente, il metodo è di tipo double, non ha parametri poiché i dati su cui lavora sono interni all’oggetto stesso. Il metodo può utilizzare gli attributi anche se questi sono privati.

Classi e metodi

```
class Rectangle
{
 int side1, side2; //lati del rettangolo
 public Rectangle(int x , int y)
        {
         side1 = x; side2 = y;
        }
 public int Area()
        {
         return (side1 * side2) ;
```



In una classe è possibile definire delle funzioni che svolgono compiti e/o calcoli

Le funzioni interne alla classe sono detti metodi. Un metodo può essere privato o pubblico.

Invocazione di metodi d’istanza

- Si utilizza la notazione puntata e il metodo è invocato a partire dall’oggetto, prima dichiarato e poi istanziato.

VC#

```
Rectangle figura = new Rectangle(3 , 7);
double risultato = figura.Area();
```

Classi e metodi

```
{
 Rectangle R = new Rectangle(4 , 5);
 x = R.Area(); //invocazione del metodo
}
```

Un metodo pubblico può essere invocato dall'esterno a partire dall'istanza

L'invocazione richiede prima il nome dell'istanza, seguito da un punto e dal nome del metodo

Focus on…

- Una classe può avere costruttori, attributi e metodi
- Gli attributi sono i dati dell'oggetto
- I metodi sono i comportamenti dell'oggetto
- Metodi e attributi possono essere sia pubblici che privati
- I metodi pubblici possono essere invocati dall'esterno
- Per invocare un metodo si scrive il nome dell'istanza, seguito dal punto, seguito dal nome del metodo (ed eventuali argomenti)

Hints

- In una buona programmazione per oggetti gli attributi di una classe saranno privati
- Si definiranno quindi metodi pubblici per l’accesso in lettura e scrittura => proprietà
- La classe fornirà poi costruttori pubblici e altri metodi pubblici per poterla rendere fruibile dall’esterno

Tipologia di metodi

- È possibile definire la visibilità coi descrittori public e private (e in seguito vedremo altre tipologie di metodi). I metodi pubblici possono essere invocati dall'esterno, mentre quelli privati sono invisibili dall'esterno.

VC#

```
public class Voto
{
   private int voto;
   private void Incrementa ( )  //---metodo Incrementa è privato
   {
     if (voto < 10)
     voto++;
   }
   public void Migliora ( int punti ) //---metodo Migliora è pubblico
   {
     for (int k = 0; k < punti; k++)
           Incrementa ();
   }
}
```

Overload di metodi

- Come per il costruttore è possibile avere più metodi con lo stesso nome ma con tipo e/o numero di parametri differente

VC#

```
public class Voto
{
   public int voto;
   public void Incrementa ( )
   {
     if (voto < 10)
       voto++;
   }
   public void Incrementa ( int punti )
   {
     for (int k = 0; k < punti; k++)
       Incrementa ();
   }
}
```

Keyword this

```
class Rectangle {
 int x, y ; //lati del rettangolo
 public Rectangle(int x)
        {
        x = x;
        y = x;
        }
public Rectangle(int x , int y)
        {
        this.x = x;
        this.y = y;
        }
}
```

Talvolta è utile specificare quandosi usano gli attributi; la parola this vuol dire QUESTA ISTANZA

Per esempio, qui si confondono i parametri con gli attributi; il parametro x copre in visibilità l'attributo

Così è semplice, nessun problema

Proprietà

- Uniscono la semplicità dei campi alla flessibilità dei metodi

- Vi si accede come campi (es. if (Persona.Eta>=18) ...) ma sono in realtà dei metodi:
  - ❑ get chiamato quando deve essere letto il valore della proprietà
  - ❑ set chiamato quando deve essere assegnato un valore alla proprietà

```
class UnaClasse
{
    private int campo = 0;
    public int UnaProprietà
    {
        get { return campo; }
        set { campo = value; }
    }
}
```

Campo

Proprietà automatiche C# 3.0

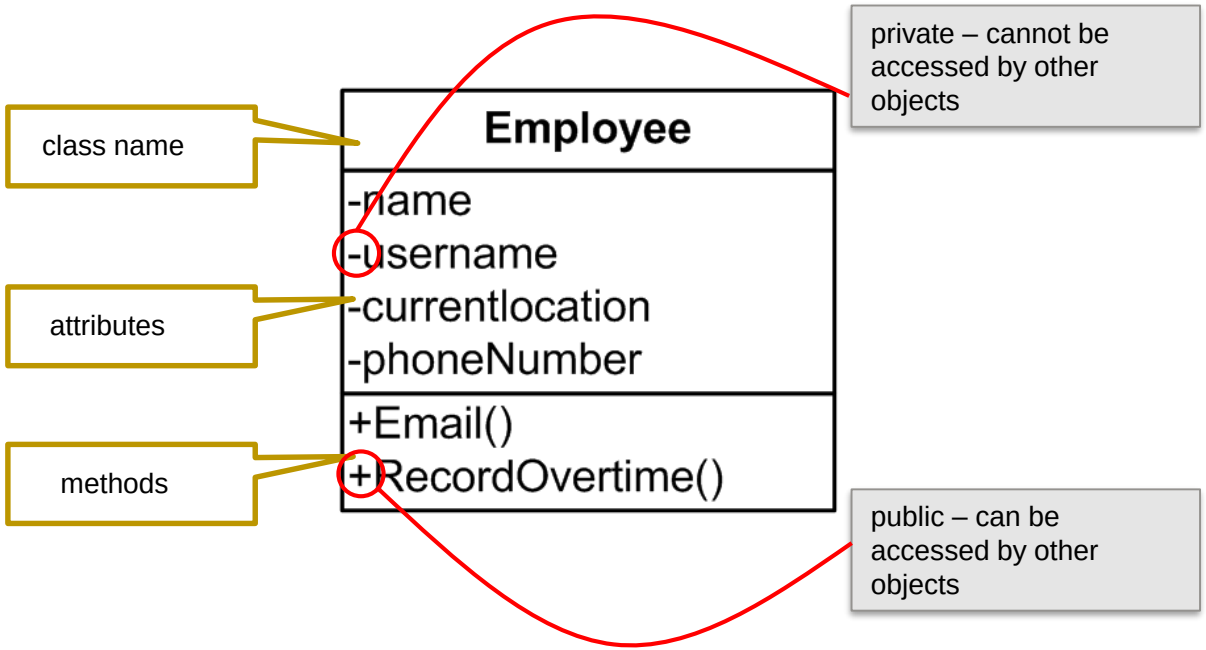
- Proprietà auto-implementate
  - ❑ Permettono di rendere la dichiarazione delle proprietà più semplice e concisa

```
class Persona
{
  public string Nome { get; set; }
  public string Cognome { get; set; }
  public DateTime DataDiNascita { get; private set; }
}
```

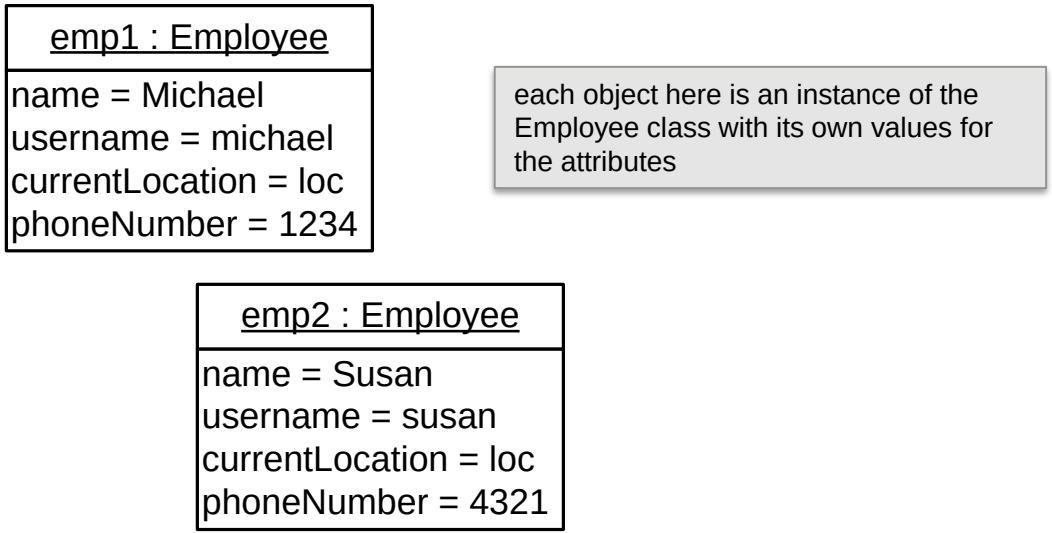
=

```
class Persona
{
    private string nome;
    private string cognome;
    private DateTime dataDiNascita;
    public string Nome
    {
       get { return nome; }
       set { nome = value; }
    }
    public string Cognome
    {
       get { return cognome; }
       set { cognome = value; }
    }
    public DateTime DataDiNascita
    {
       get { return dataDiNascita; }
    }
}
```

Class diagram - UML



Object diagram - UML



Auto implemented properties

- https://www.youtube.com/watch?v=Z70AwnYYJOc (2:40 min)
- After the video answer to the following questions:

1. What is a property?
2. What is an auto impemented property?
3. When can we use the auto implemented property?

Example with properties

```
class Example
{
    int _number;
    public int Number
    {
        get
        {
            return this._number;
        }
        set
        {
            this._number = value;
        }
    }
}
```

```
static void Main()
{
    Example example = new Example();
    example.Number = 5; // set { }
    Console.WriteLine(example.Number); // get { }
}
```

Example with auto implemented properties

```
class Example
{
    public int Number
    {
        get;
        set;
    }
}
```

```
static void Main()
{
    Example example = new Example();
    example.Number = 8;
    example.Number *= 4;
    Console.WriteLine(example.Number);
}
```

Example with automatic default value

```
class Medication
{
    public int Quantity { get; set; } = 30; // Has default value.
}
```

```
static void Main()
{
    Medication med = new Medication();
    // The quantity is by default 30.
    Console.WriteLine(med.Quantity);
    // We can change the quantity.
    med.Quantity *= 2;
    Console.WriteLine(med.Quantity);
}
```

Example with read only properties

```
private readonly int birthYear = 1998;
```

- Here the birthYear field is given a value of 1998 and cannot be changed to a different value elsewhere in the class without generating a compiler error.

Example with init only properties

- Init-only setters are a newer language feature that give you the ability to set read-only properties of a class at construction without needing to add constructor parameters.

```
public class Pet
{
   public Pet(string breed, int birthYear, string name)
   {
   }

   // Get-only auto properties
   public string Breed {get;}
   public int BirthYear {get;}
   public string Name {get;}
}
```

Here we can create a new `Pet` with code like the following:

```
Pet myPet = new Pet("Cairn Terrier", 2016, "Jester");
```

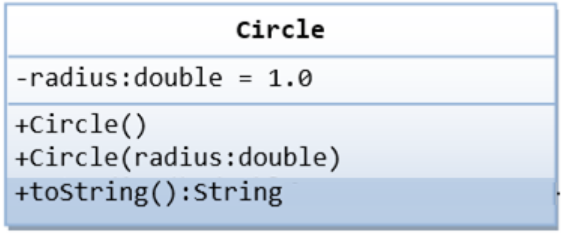
```
public class Pet
{
   public string Breed {get; init;}
   public int BirthYear {get; init;}
   public string Name {get; set;}
}
```

Note that this class doesn't declare a constructor beyond the default empty constructor. Admittedly this is a very simple class that perhaps should be a `stuct` or a `record`, but simple is fine for articles.

We could create an instance of `Pet` with the following code that provides an initializer after the constructor call:

```
Pet myPet = new Pet()
{
   Breed = "Cairn Terrier",
   BirthYear = 2016,
   Name = "Jester"
};
```

Exercise



A class called circle is designed as shown in the following class diagram. It contains:

•Two private instance variables: radius (of the type double) and color (of the type string), with default value of 1.0 and "red", respectively.

•Two overloaded constructors – a default constructor with no argument, and a constructor which takes a double argument for radius. Check the values when setting in the constructor

•Two properties: Radius and Area, which return the radius and area of this instance, respectively.

•Modify the class Circle to include a third constructor for constructing a Circle instance with two arguments – a double for radius and a String for color.

•Add the methods Area() and Circumference()

Lesson closure: Explain a Procedure

- Create a map about the properties