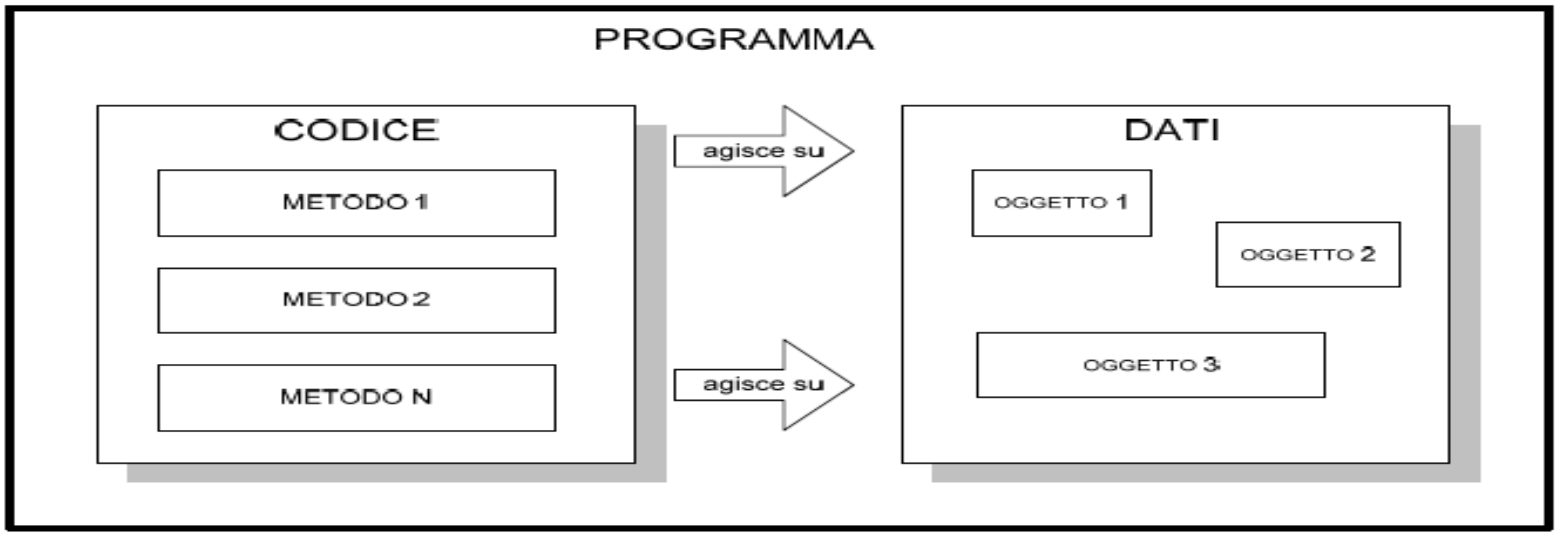
OOP
Le classi

Informatica
Prof. Viglietti Francesco
a.s. 2012-13

OOP 1

La programmazione ad oggetti: scoprire le sue potenzialità rispetto alla programmazione procedurale. La situazione di partenza è quella riassunta nello schema seguente:



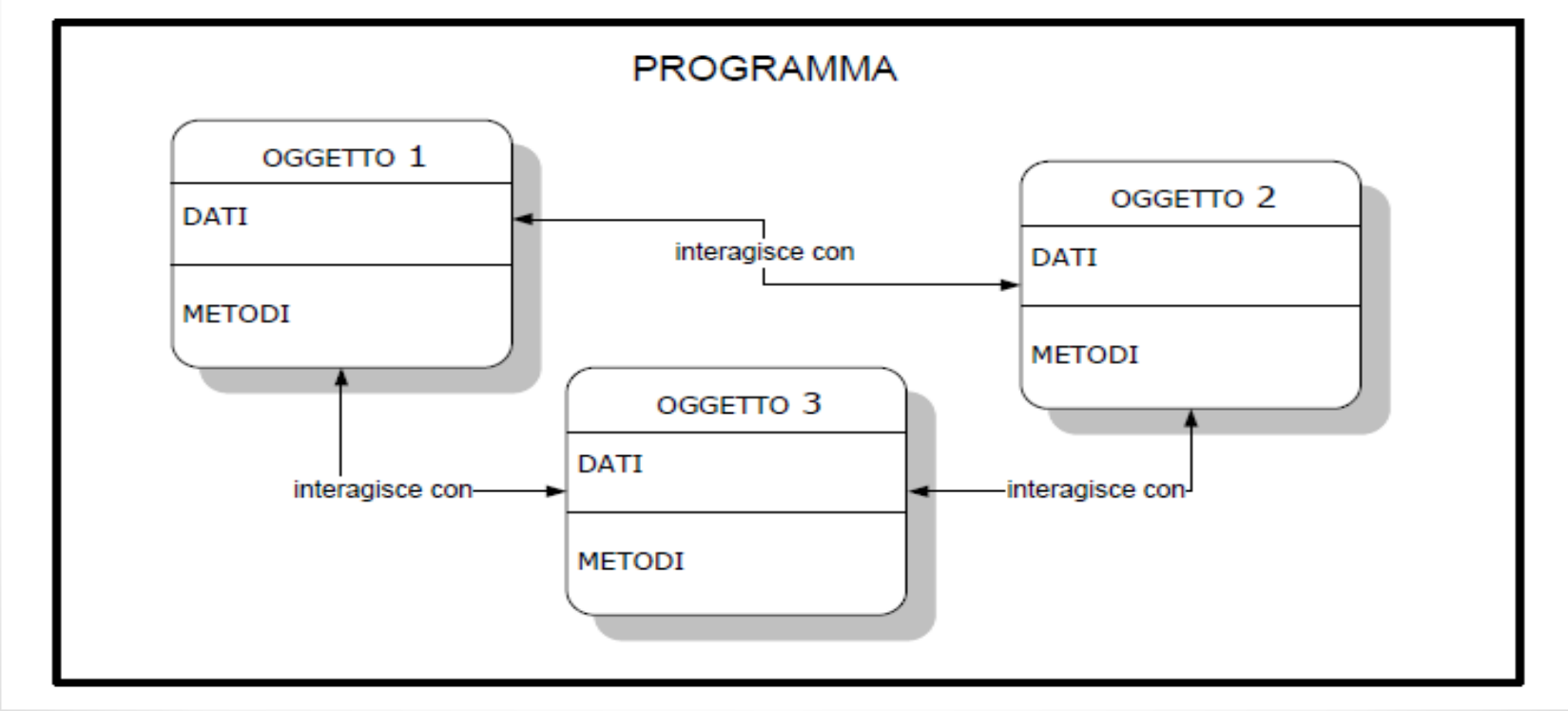
Si può riassumere dicendo che un programma (codice e dati) può essere suddiviso in 2 parti. La parte del codice, che a sua volta comprende vari metodi in cui il problema viene scomposto per la sua risoluzione, e la parte di dati che fa riferimento alla presenza di più oggetti (variabili o altri tipi di dato). Queste 2 parti sono distinte pur entrando in contatto tra loro durante l'esecuzione del programma, in particolare sarà il codice (insieme di istruzioni) ad agire sui dati per produrre un certo risultato.

OOP 2

Nella programmazione ad oggetti questo aspetto assume una rappresentazione diversa. Iniziamo a definire alcuni aspetti della OOP:

- I dati non sono separati dal codice che li elabora
- Cambia il ruolo di un oggetto che non sarà solo un contenitore di dati, ma ***fornisce sia una rappresentazione per i dati, sia l'insieme delle operazioni che possono essere eseguite su di essi.***

Chiariamo il concetto con una rappresentazione grafica:



Definizioni 1

Si nota come gli oggetti siano degli elementi indipendenti tra loro ma che interagiscono. Al loro interno contengono sia dati che metodi.

Il **C#** è un linguaggio orientato agli oggetti (**O.O.P.)**.

Un **oggetto** è la realizzazione in pratica di un qualcosa descritto in teoria. Esso possiede caratteristiche, proprietà, attributi e metodi per utilizzarlo.

Quando un architetto progetta una casa, fa il progetto su carta descrivendone dimensioni, caratteristiche ed eventualmente le tecniche di costruzione. Questo lavoro rappresenta la descrizione teorica di un oggetto (Casa) che dovrà essere costruito. In questo esempio, il progetto rappresenta la **classe**, mentre la casa realmente costruita, rappresenta l'**oggetto**.

Usando la terminologia ad hoc, si dice che è stato **istanziato l'oggetto** casa cioè è stata creata una **istanza della classe** (progetto) Casa. In parole povere le classi sono definizioni usate per la creazione di oggetti. Dal progetto dell'architetto, si possono costruire più case.

Definizioni 2

Per le classi, il concetto è lo stesso; da una **classe** si possono **istanziare** più **oggetti**, dipende da quanti oggetti dello stesso tipo abbiamo bisogno in una applicazione. Ad esempio:

la **classe** Studente descrive tutte le caratteristiche di uno studente, per es. il nome, il cognome, la provenienza, ecc.

Con una **istanza** della **classe** Studente, che chiameremo **Alunno**, creeremo effettivamente un **oggetto** reale a partire della **classe** e che potremo usare nel nostro programma. E' ovvio che in una aula di una scuola c'è più di un alunno, quindi istanzieremo più **classi** Studente per formare la nostra aula di oggetti **Alunno**.

Incapsulamento

Le caratteristiche principali di un linguaggio O.O.P. sono:

l'incapsulamento, il polimorfismo, l'ereditarietà.

L'incapsulamento è un concetto tramite il quale una **classe** consente di incorporare dati e tutti i metodi (funzioni) che li gestiscono. Ha la possibilità di nascondere questi elementi interni al mondo esterno. Le **classi** hanno dei meccanismi tali da rendere fruibili questi dati senza che l'utente sia a conoscenza di come sono strutturati. Nella maggior parte dei casi, è cosa positiva non rendere pubblici i dati di una classe perché un utente (programmatore) deve solamente utilizzarla, non sapere cosa e come i dati sono stati impostati. Il codice del programma non sarà in grado di accedere ai dati della classe. Una classe avrà al suo interno dei metodi (li vedremo più avanti) per renderli disponibili. Cosi facendo, il programmatore della classe potrà modificare liberamente la struttura senza che il codice del programma ne subisca conseguenze.

Polimorfismo

Il **polimorfismo** è la capacità di assumere molte forme. Ad es., abbiamo fatto uso di un metodo per stampare sul monitor: Console.WriteLine(".....{0}",var); . L'argomento tra parentesi graffe {0}, si riferisce alla variabile var che dovrà essere stampata. Ma il metodo non sa di che tipo è la variabile che deve stampare. Il metodo in questione è polimorfico cioè si adatta alla maggior parte dei tipi che gli vengono passati. Lo stesso metodo può avere più di una variabile da stampare e si adatta.

Ereditarietà

L'ereditarietà è la possibilità di creare una **classe** che eredita le caratteristiche da un'altra **classe** espandendone le funzionalità. Per es. possiamo creare una classe generale che chiameremo Autoveicoli che raggruppa tutti i veicoli che hanno in comune alcune proprietà e funzionalità. Possiamo creare una classe Autovettura che eredita le proprietà e funzionalità comuni e in più possiamo aggiungere proprietà e funzionalità tipiche di un'autovettura, per esempio marca, modello, motorizzazione ecc... . Abbiamo quindi creato una classe che ha ereditato le funzionalità dalla classe "genitore" e ne abbiamo espanso le funzionalità aggiungendone di nuove non presenti nella classe "genitore". L'ereditarietà è un concetto molto potente che ci consente di creare oggetti (classi) molto complessi e pieni di funzionalità.

Definizione di classe

Per definire una classe, abbiamo bisogno di una intestazione rappresentata dalla parola chiave **class** e da un corpo che ne racchiude il contenuto, così:

class *NomeClasse*

{ *elementi definiti nel corpo* **}**

dove il ***NomeClasse*** o ***identificatore*** rappresenta il nome che vogliamo dare alla classe. E' utile dare un nome significativo alla classe che ne descriva l'impiego. Ad esempio possiamo definire la classe Studente:

class *Studente*

{

elementi definiti nel corpo

}

dal suo identificatore (*Studente*), capiamo al volo che si tratta di qualcosa che a che fare con una persona; un altro esempio:

class *Autovettura*

{

elementi definiti nel corpo

}

anche in questo caso si capisce, dall'identificatore (Autovettura), che avremo a che fare con un veicolo.

Istanziazione e proprietà

Una volta creata una classe, non possiamo utilizzarla senza dichiararla. Siamo di fronte ad un tipo particolare di dato che noi abbiamo realizzato (progetto) e da questo dobbiamo creare un **oggetto** che lo rappresenti e che possiamo utilizzare. La dichiarazione (**istanza**) di una classe avviene in maniera quasi banale, cioè faremo così:

NomeClasse *nomeOggetto* = **new NomeClasse()**;

ritornando agli esempi di prima, possiamo **istanziare** un oggetto **Studente** che chiameremo **Alunno** o un oggetto **Autovettura** che chiameremo **MiaMacchina** così:

Studente *Alunno* = **new Studente ()**;

Autovettura *MiaMacchina* = **new Autovettura ()**;

abbiamo creato due oggetti, Alunno di tipo **Studente** e MiaMacchina di tipo **Autovettura**.

Ma questi oggetti non servono a molto in una applicazione, sono oggetti "vuoti", cioè non contengono al loro interno qualcosa che li faccia distinguere tra loro. Per esempio un Alunno avrà pure un nome ed un cognome, la MiaMacchina avrà pure una cilindrata, una marca, un colore, ecc. . Ebbene, per avere queste informazioni, dobbiamo necessariamente inserire all'interno del loro corpo delle variabili che ci indicano tali informazioni, o per meglio dire, tali **proprietà**.

Classe studente

Tali variabili vengono definite, **variabili** o **campi istanza**. Facciamo un esempio di come vengono dichiarati i campi istanza. Prendiamo la classe **Studente**, possiamo affermare che uno studente ha un nome ed un cognome che potremmo rappresentare tramite due variabili di tipo stringa. Ecco come:

```
class Studente
    {       // definisco due campi istanza
       string nome, cognome;
    }
```

Così com'è, non è utilizzabile, pur essendo una classe a tutti gli effetti, infatti, se istanziassimo un oggetto Alunno, non potremo mai sapere come si chiama. I suoi campi istanza non saranno fruibili dal codice esterno perché come sono stati dichiarati diventano automaticamente privati (**private**) alla classe e nessun codice può accedervi. Qui entrano in gioco i modificatori di accesso. Dobbiamo fare in modo che questi campi istanza diventino **pubblici**, cioè accessibili a tutto il codice utente. Questa è la nuova classe:

```
class Studente
 {          // definisco due campi istanza
     public string nome;
     public string cognome;
 }
```

Classe studente 2

Finalmente ora la nostra classe è pronta. Abbiamo inserito il modificatore **public** con l'intenzione di utilizzare i campi istanza all'esterno della classe e quindi dal nostro programma. Come vedremo più avanti, i campi istanza **private** potranno essere comunque manipolati tramite un costrutto molto semplice. Però adesso concentriamoci sulla nostra classe e vediamo come valorizzare questi campi.

Per prima cosa dovremo istanziare un **oggetto** dalla classe tramite il quale accederemo ai campi istanza che abbiamo creato, quindi:

Studente Alunno=new **Studente**();

Per accedere ai campi istanza faremo uso di una particolare notazione, la notazione puntata in questo modo:

Alunno.nome="Mario";

con questa istruzione, abbiamo assegnato al campo istanza **nome** dell'oggetto **Alunno** la stringa "Mario".

Allo stesso modo faremo per il cognome:

Alunno.cognome="Rossi";

Classe alunno

All'assegnazione di un valore ad una variabile, ne corrisponde anche una lettura e nel nostro caso faremo l'operazione inversa; se avessimo una variabile stringa chiamata per esempio *datoNome*, potremo valorizzarla leggendo il valore memorizzato nel campo istanza del nostro oggetto:

datoNome=**Alunno**.nome;

i campi istanza sono delle normali variabili e quindi le possiamo manipolare come sappiamo.

A lato vediamo il codice per la classe Alunno.

N.R	Codice sorgente Alunno.cs
1	using System; // questo programma usa la classe Studente
2	class Studente // la classe
3	{
4	public string nome; // campi istanza della classe
5	public string cognome;
6	}
7	class Alunno // il programma
8	{
9	public static void Main()
10	{
11	// istanzio la classe
12	Studente Alunno = new Studente();
13	// valorizzo i campi dell'oggetto
14	Alunno.nome = "Mario";
15	Alunno.cognome = "Rossi";
16	// stampo i valori dei campi
17	Console.WriteLine("\nIl nome dell'alunno è {0} {1}",
18	Alunno.nome,Alunno.cognome);
19	// attendo un carattere per chiudere il programma
20	Console.ReadKey();
21	}
22	}

Debolezze

Questo esempio ha svariate debolezze. La più evidente è quella di presentare tutti i membri come pubblici. In generale questa impostazione rende in un certo senso vulnerabile la classe e se questo non è particolarmente grave in un banale programma stand alone come l'esempio precedente può essere fonte di problemi in progetti più grossi dato che si possono avere accessi incontrollati da qualunque parte ai membri interni della classe. Vale la pena abituarsi subito ad altre tecniche ... Va aggiunto, per quanto dovrebbe essere sottointeso, che una classe al suo interno può contenere qualsiasi tipologia di dati.

Facciamo comunque ora un piccolo passo in avanti…

Costruttore

```
using System;
class PuntoSulPiano
{
    public PuntoSulPiano(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public int x;
    public int y;
}

class program
{
    public static void Main()
    {
        PuntoSulPiano p1 = new
PuntoSulPiano(1,2);
    }
}
```

La riga in rosso ci presenta un metodo speciale per le classi (applicabile anche alle strutture) che si chiama costruttore. Tecnicamente questo altro non è che un membro che implementa le azioni necessarie alla inizializzazone di una istanza di classe. Un costruttore deve avere lo stesso nome della classe a cui appartiene e viene eseguito immediatamente in fase di creazione dell'oggetto. E' quello che succede nell'esempio.
Un costruttore comunque non necessariamente inizializza delle variabili, è anche ammissibile costruire una classe.
E' possibile anche non definire un costruttore nel qual caso sarà il compilatore a fornirne uno che inizializzerà i dati ai rispettivi valori di default. In ogni caso quindi un costruttore, definito dall'utente o di default, c'è sempre.

Esempi …

esempio1

```
using System;
class PuntoSulPiano
{
    public int x;
    public int y;
}
class program
{
    public static void Main()
    {
        PuntoSulPiano p1 = new PuntoSulPiano();
        Console.WriteLine(p1.x);
    }
}
```

La classe definita, non ha alcun costruttore per cui, come accennato, viene usato quello di default che attribuirà il valore 0 a x e a y. Il compilatore si farà sentire segnalando che le due variabili, non essendo state assegnate, avranno valore 0.

Esempi …

esempio2

```
class PuntoSulPiano
{
    public PuntoSulPiano()
    {
        x = 1;  y = 3;
    }
    public int x; public int y;
}
```

Quando viene creata l'istanza di classe entra in funzione il costruttore assegnando i valori che sono prefissati all'interno della classe.

esempio3

```
class PuntoSulPiano
{
    public PuntoSulPiano(int x, int y)
    {
        this.x = x; this.y = y;
    }
    private int x; private int y;
}
```

Ora il costruttore accetta dei parametri dall'esterno per cui non necessariamente ogni istanza avrà membri con lo stesso valore iniziale come nel passaggio precedente.

Da notare nell'esempio 3 che x e y sono private quindi protette rispetto all'esterno.

Costruttori statici

Vengono usati di norma o per inizializzare dei dati statici o per eseguire operazioni che devono essere compiute una sola volta, è più o meno la definizione che si trova un po' ovunque. I costruttori statici sono soggetti ad una serie di regole precise in particolare:

- non possono essere invocati direttamente, quindi non se ne può controllare l'esatto attimo di entrata in gioco

- non accettano modificatori di accesso e nemmeno parametri.

Piuttosto rigidi, se vogliamo, comunque anche questi vengono chiamati in fase di start up della classe. Interessante notare che la presenza di un costruttore statico non impedisce che la classe ne abbia anche uno dinamico.

Costruttori… esempi

```
    //esempio 1
1   using System;
2   class Saluto
3   {
4       static Saluto()
5       {
6           Console.WriteLine("Hello");
7       }
8       public Saluto()
9       {
10          Console.WriteLine("Ciao");
12      }
13  }
14  class program
15  {
16      public static void Main()
17      {
18          Saluto s1 = new Saluto();
19          Console.ReadLine();
20      }
21  }
```

```
1   using System;
2   class Punto
3   {
4       public Punto(int a, int b, int c)
5       {
6           this.a = a; this.b = b; this.c = c;
7       }
8       public Punto(int x, int y)
9       {
10          this.x = x; this.y = y;
12      }
13      public int x; public int y;
14      public int a; public int b;
15      public int c;
16  }
17  public class Class1
18  {
19      public static void Main()
20      {
21          Punto p1 = new Punto(1,2,3);
22          Punto p2 = new Punto(1,2);
23          Console.WriteLine(p2.x);
24      }
25  }
```