La metodologia Object Oriented

Metodologia OO: paradigmi

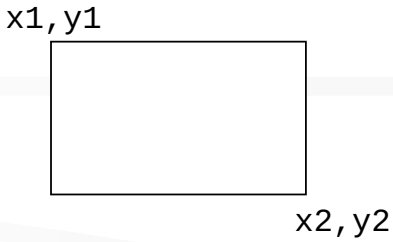
- Paradigmi di programmazione:
  - Programmazione procedurale
  - Programmazione modulare
  - Astrazione sui dati
  - Programmazione Object Oriented
- Un paradigma può teoricamente essere seguito utilizzando qualsiasi linguaggio di programmazione.

Programmazione Procedurale

Si definiscano le procedure desiderate; si utilizzino gli algoritmi migliori.

- L'attenzione è posta sugli algoritmi
- I dati subiscono *passivamente* le azioni dell'algoritmo

P P: esempio 1

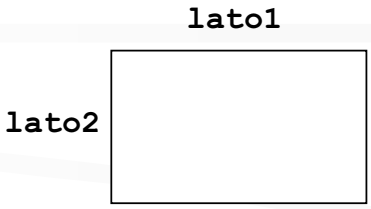



```
typedef struct {
 int x1, y1;
 int x2, y2;
} Rettangolo;

int areaRettangolo(Rettangolo r) {
 return (r.x2-r.x1)*(r.y2-r.y1);
}

int main()
{ Rettangolo r;    // Uso del type rettangolo
  r.x1=2; r.y1=2;
  r.x2=9; r.y2=10;
  printf("Area rettangolo=%d\n",
          areaRettangolo(r));
}
```

P P: esempio 2




```
#include <stdio.h>
typedef struct {
 int lato1;
 int lato2;
} Rettangolo;

int Area(Rettangolo r){return (r.lato1*r.lato2);}

int main() {
 Rettangolo r;
 r.lato1=5; r.lato2=6; //Modifica necessaria!
 printf("Area = %d,Area(r));
}
```

P P: esempio 3

```
#include <stdio.h>
 typedef struct {
    int lato1;
    int lato2;
  } Rettangolo;

 int Area(Rettangolo r){return (r.lato1*r.lato2);}
void Init(Rettangolo* r, int l1, int l2){r->lato1=l1;
                                         r->lato2=l2; }

 int main() {
  Rettangolo r;
  Init(&r, 3, 5);
 printf("Area = %d”,Area(r));
 printf(”Lato1 = %d”,r.lato1); // Accesso diretto a lato1
 }
```

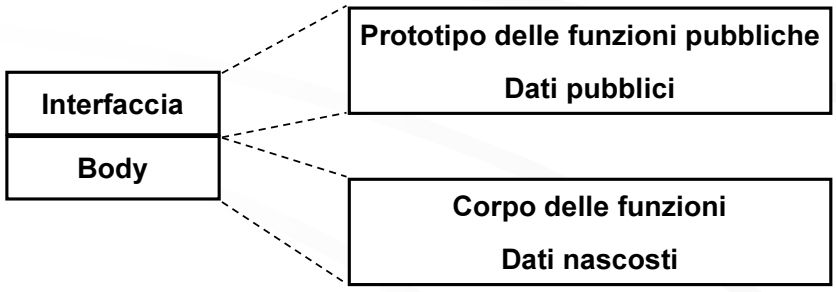
Programmazione Modulare

Si decida quali moduli si desiderano; si suddivida il programma in modo che i dati siano nascosti nei moduli.

- Utile dove esiste un insieme di procedure con dati associati
- Maggiore attenzione ai dati: funzioni come mezzo di accesso ai dati
- Presenza di un'interfaccia

Programmazione Modulare...

- Separazione Interfaccia/Realizzazione



PM : Esempio

```
DEFINITION MODULE Rettangolo
EXPORT Init,Area

PROCEDURE Init(px1,py1,px2,py2:INTEGER)
PROCEDURE Area():INTEGER

END Rettangolo
```

PM : Esempio ...

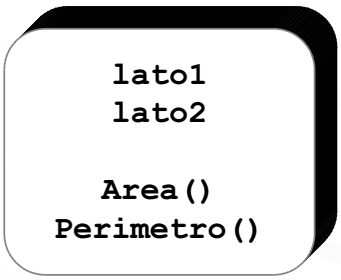
```
IMPLEMENTATION MODULE Rettangolo
VAR x1,y1,x2,y2:INTEGER
PROCEDURE
Init(px1,py1,px2,py2:INTEGER)
 BEGIN
    x1=px1
    y1=py1
    x2=px2
    y2=py2
 END
PROCEDURE Area():INTEGER
 BEGIN
   RETURN ((x2-x1)*(y2-y1))
 END
END Rettangolo
```

PM : Esempio ...

```
MODULE main
FROM Rettangolo IMPORT Init,Area
VAR area:INTEGER
BEGIN
  Init(2,2,9,10);
  area=Area();
  WRITE("L'area e' pari a ...", area);
  Init(5,5,8,1); Altro rettangolo!!
  area=Area();
  WRITE("L'area e' pari a ...", area);
END
END main
```

ADT - Abstract Data Type

- Dati e operazioni come unica entità



ADT - Abstract Data Type

Si decida quali tipi si desiderano

Si fornisca un insieme completo di operazioni per ogni tipo

ADT - Abstract Data Type

- **I nuovi tipi si comportano (quasi) nello stesso modo di quelli predefiniti**
- **Una ADT differisce da un modulo in quanto è un tipo di dati da cui ottenere più variabili (oggetti) utilizzabili come un modulo**.

ADT - Abstract Data Type

```
#include <stdio.h>
typedef struct {
 int lato1;
 int lato2;
 void init(int l1, l2){ lato1=l1; lato2=l2;}
 int Area() {return lato1*lato2;}
 int Perimetro() {return 2*(lato1+lato2);}
} Rettangolo;

int main() {
 Rettangolo r;
 r.init(5,6);
 r.lato1=7;  // Permessa ma non conveniente!!
 printf("Area = %d  Perimetro = %d",
         r.Area(), r.Perimetro());
}
```

Programmazione Orientata agli Oggetti

Si determini quali classi si desiderano

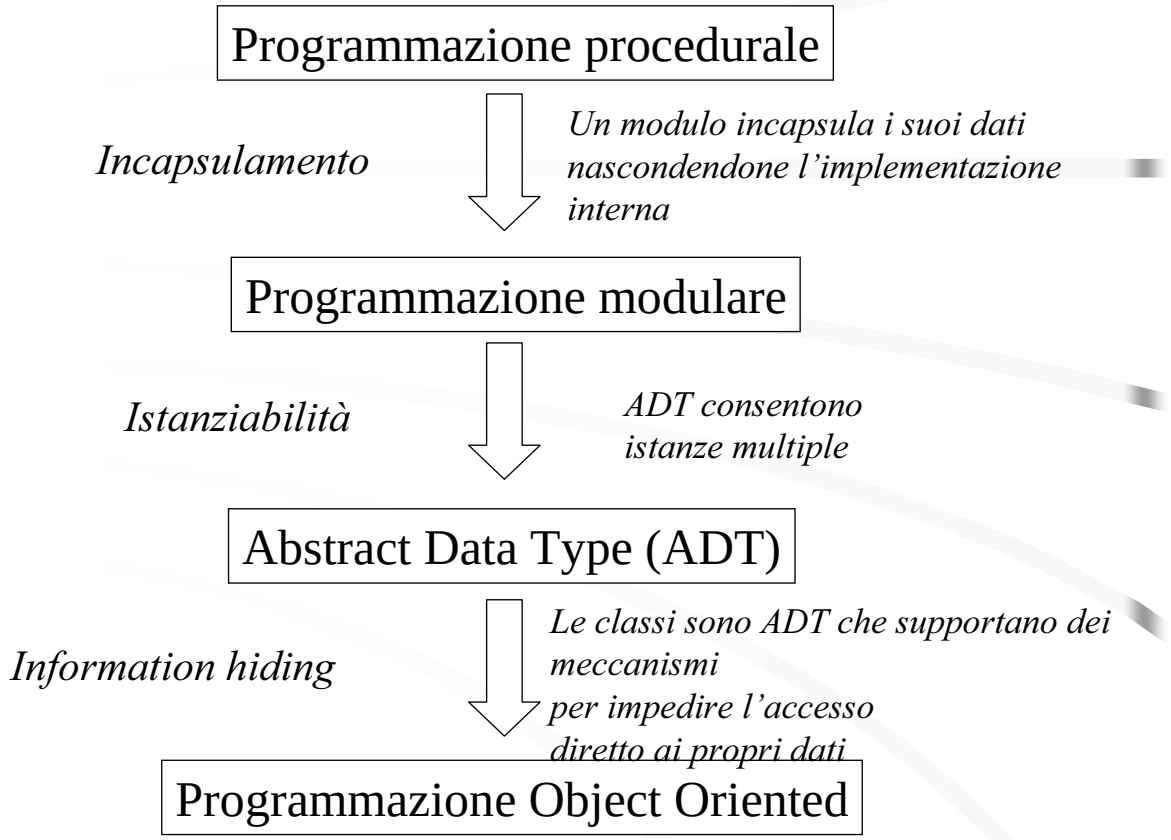
↓

Si fornisca un insieme completo di operazioni per ogni classe

↓

Si renda esplicito ciò che hanno in comune mediante l'ereditarietà

(programmazione alla differenze)



Object Oriented

- **I sistemi *orientati agli oggetti* sono caratterizzati dai concetti di:**
  - oggetti e identità
  - classi
  - associazione tra classi
  - ereditarietà tra classi
  - polimorfismo
  - parametrizzazione di tipi e funzioni

OO - Oggetti

- **Ogni entità del sistema che si vuole descrivere è modellata attraverso il concetto di oggetto**

- **Gli oggetti possono essere sia concreti che concettuali**
  - un esempio di oggetto concreto: un file in un file system
  - un esempio di oggetto concettuale: una politica di scheduling in un sistema operativo multiprocessing

OO - Oggetti

- Un **oggetto** è un'entità software con *stato*, *comportamento* e *identità*
  - lo stato è modellato da un insieme di variabili locali (attributi);
  - il comportamento è modellato da un insieme di procedure locali chiamate *metodi* o *funzioni membro*;
  - l'identità è immutabile, indipendente dal valore dello stato, e rende un oggetto diverso da ogni altro.

OO - Oggetti

- Le operazioni applicabili ad un oggetto sono dette ***messaggi*** a cui l'oggetto risponde, restituendo un valore memorizzato nello stato o calcolato con una procedura locale
- L'insieme dei messaggi a cui un oggetto può rispondere è chiamato ***interfaccia*** dell'oggetto

OO - Oggetti

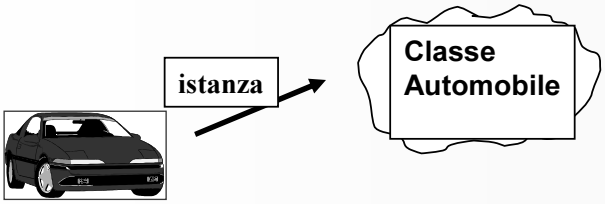
- Si dice che un oggetto ***incapsula*** lo stato quando ne consente l'accesso solo tramite i propri metodi

OO - Classi

- Una classe descrive un insieme di oggetti con proprietà simili (attributi) e comportamento comune (metodi)
- Una classe è un'astrazione che descrive le proprietà rilevanti di una data applicazione

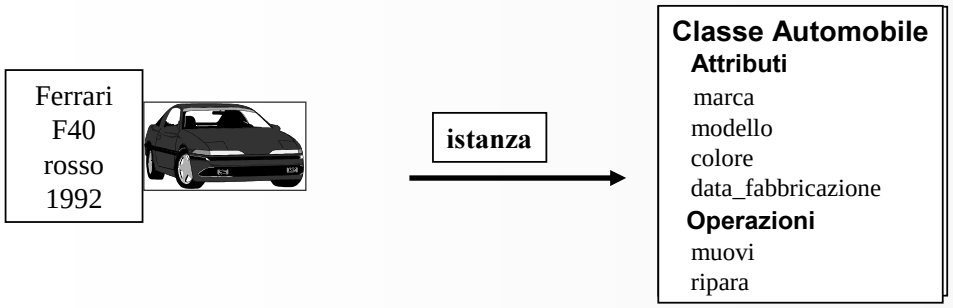
OO - Classi

- La scelta delle classi è arbitraria e dipende dalle applicazioni che si considerano
- Gli oggetti appartenenti a una classe sono detti istanze



OO - Classi

- Ogni oggetto istanza di una classe ha propri valori per gli attributi definiti nella propria classe, ma condivide con le altre istanze della stessa classe i nomi degli attributi e le operazioni



OO - Attributi

- Un attributo descrive una proprietà di un oggetto
  - Nome, età, nazionalità sono attributi della classe Persona
  - Marca, modello, colore sono attributi della classe Automobile
  - Ogni attributo ha un valore per ogni oggetto

OO - Attributi: esempio

Joe Smith
25
U.S.A.

Toro Seduto
35
America

Onama Oti
25
Polinesia

Classe Persona
Attributi
nome
età
nazionalità
Operazioni
mangia
parla

STATO

COMPORTAMENTO

OO - Metodi

- Un metodo rappresenta un'azione o trasformazione che un oggetto esegue o a cui è sottoposto
  - Esempi:
    - muovi per la classe Automobile
    - mangia per la classe Persona
    - open, close, hide, per la classe Window
- Tutti gli oggetti di una classe condividono le stesse operazioni

OO - Metodi

- Il comportamento dell'operazione dipende dalla classe di appartenenza dell'oggetto
- Ogni oggetto "conosce" la sua classe di appartenenza e quindi la giusta implementazione dell'operazione

OO - Metodi

- La stessa operazione può essere invocata su oggetti di classi diverse; tale operazione è detta ***polimorfa***
  - Esempio. Classe *Persona* e metodo *presentati*:

    A seconda delle caratteristiche di una persona (studente, impiegato ecc.) il metodo presentati assume significati diversi.

OO - Associazioni

- Un'***associazione*** fra due classi A e B è una relazione binaria che associa oggetti di A con oggetti di B:
  - *lavora per* che modella il fatto che un impiegato lavora per una data azienda.
  - la relazione inversa *ha impiegati* modella il fatto che un'azienda ha un certo numero di impiegati.

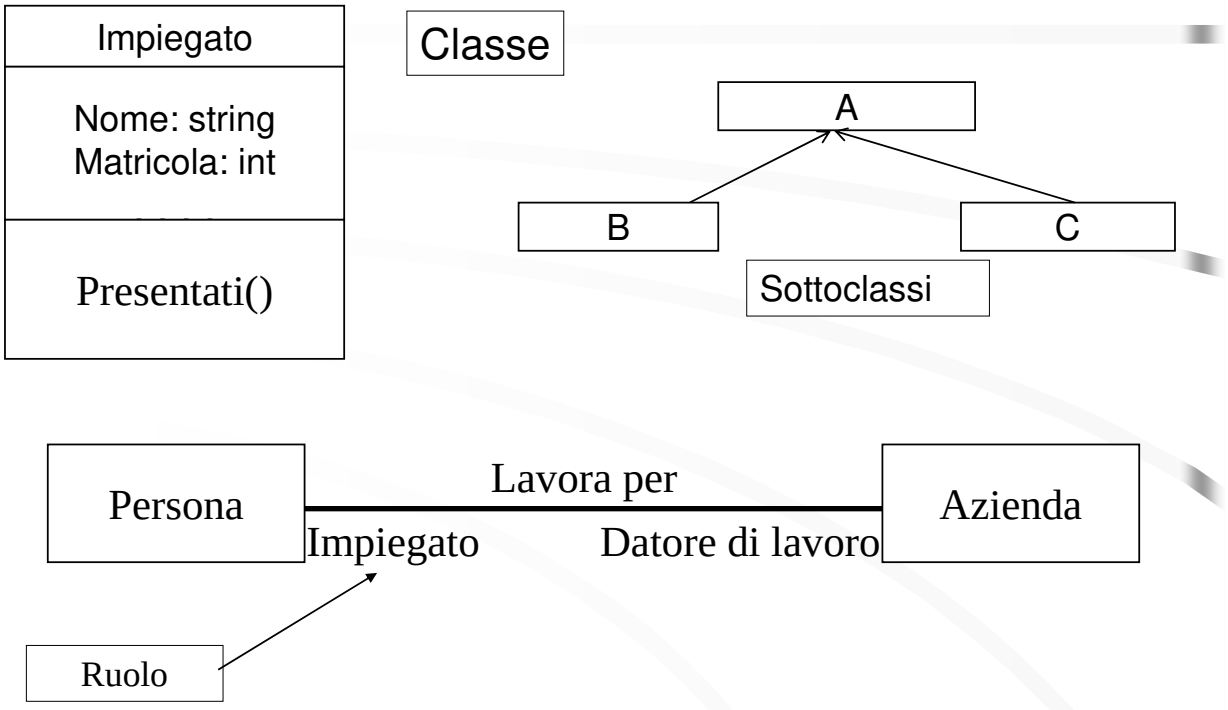
OO - Associazioni

- M**olteplicità** di un'associazione:
  - fra due classi A e B specifica il numero di istanze di B che possono essere correlate a una singola istanza della classe A
- Classificazione:
  - uno a uno
  - uno a molti
  - molti a uno
  - molti a molti

OO - Associazioni

- **uno a uno** quando ad un oggetto di A corrisponde un solo aggetto di B e viceversa;
- **uno a molti** quando ad un oggetto di A corrispondono più oggetti di B, ma ad un oggetto di B può corrispondere un solo oggetto di A;
- **molti a uno** quando ad un oggetto di A corrisponde un solo aggetto di B, ma ad un oggetto di B possono corrispondere più oggetti di A;
- **molti a molti** quando ad un oggetto di A corrispondono più oggetti di B e viceversa.

OO - Notazione grafica

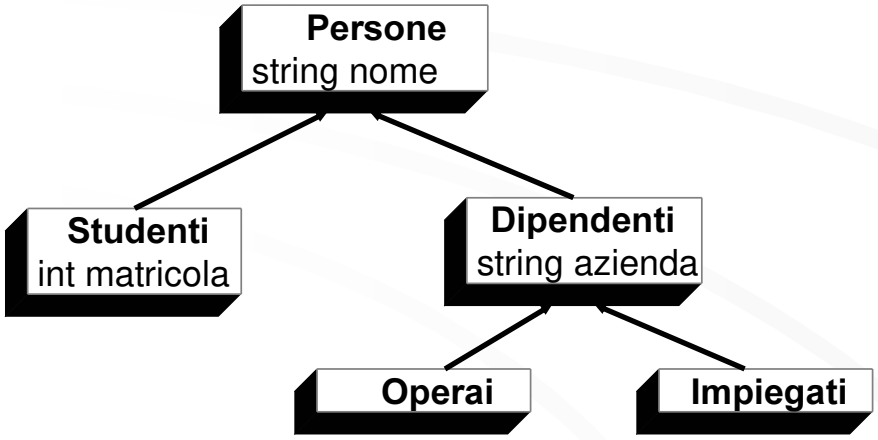


OO - Ereditarietà

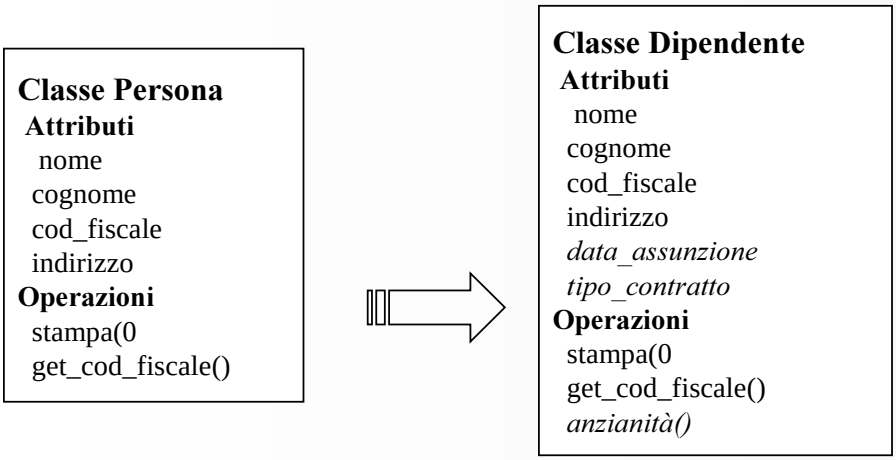
- Il meccanismo dell'ereditarietà consente:
  - La definizione di nuove classi a partire da classi esistenti specificando rispetto a queste solo la parte aggiuntiva (**riutilizzo** del software)
  - L'analisi e la rappresentazione di una certa realtà attraverso specializzazioni successive.

OO - Ereditarietà

- Le **gerarchie** fra classi (*generalizzazione ed ereditarietà*) sono potenti astrazioni per condividere similitudini tra classi



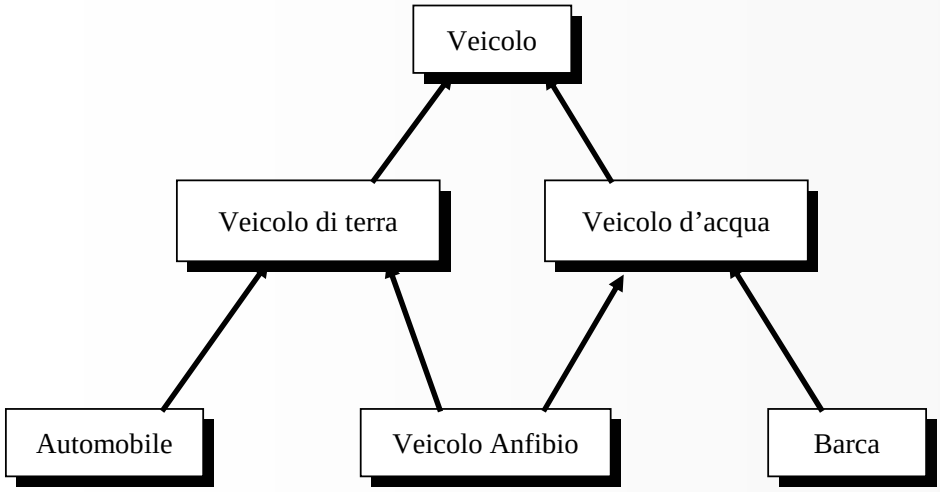
OO - Ereditarietà: esempio



OO - Ereditarietà multipla

- Permette ad una classe di avere più di una superclasse "diretta" e di ereditare da tutte
  - Vantaggi:
    - mescolare le informazioni di due o più sorgenti
    - maggior potenza nella specifica delle classi
    - maggior possibilità di riutilizzo
  - Svantaggi
    - perdita di semplicità concettuale
    - difficoltà di implementazione

OO - Ereditarietà multipla



OO - Classi astratte

- Una classe ***astratta*** è una classe che non ha istanze proprie ma ha solo istanze dovute alle sue sottoclassi
- Una classe ***concreta*** è una classe istanziabile cioè può avere istanze dirette
- Le classi astratte sono spesso usate per definire le interfacce alle gerarchie