La programmazione orientata agli oggetti

Simone Campagna

Paradigmi di programmazione

- Logica
- Funzionale
- Procedurale
- Strutturata
- Orientata agli oggetti
- Orientata agli eventi
- Orientata agli aspetti
- ...

Paradigmi di programmazione/2

Non c'è un paradigma che sia in assoluto migliore degli altri, dipende in buona misura dalla natura del problema. Alcuni paradigmi si applicano ad ampie classi di problemi.

I linguaggi possono offrire supporto ad uno o più paradigmi. Ad esempio, Smalltalk ed Eiffel supportano SOLO il paradigma OO.

I linguaggi più usati (C++, python) offrono supporto a vari paradigmi.

Visual Basic non è orientato agli oggetti; Visual Basic .NET fornisce supporto per OOP.

Perché nasce l'esigenza dell'approccio OO

Il punto di forza del paradigma OO è costituito dalla capacità di gestione della complessità.

La complessità del software è una caratteristica *essenziale,* e non accidentale, per quattro ragioni:

- Complessità del dominio del problema
- Difficoltà di gestione del processo di sviluppo
- L'esigenza di flessibilità
- Il problema della descrizione di sistemi discreti

Gli attributi di un sistema complesso

- Composto da una gerarchia di sottosistemi di minore complessità
- La scelta delle componenti primitive non è assoluta, ma dipende dall'osservatore del sistema
- Le relazioni fra le parti interne degli oggetti sono più forti delle relazioni fra oggetti
- I sottosistemi sono normalmente di pochi tipi diversi (presenza di pattern comuni)
- Sono generalmente evoluzioni di sistemi più semplici

OOP

La programmazione orientata agli oggetti (OOP) è una tecnica di implementazione consistente nell'organizzare i programmi come insiemi interagenti di oggetti.

Gli oggetti sono istanze di classi che formano una gerarchia.

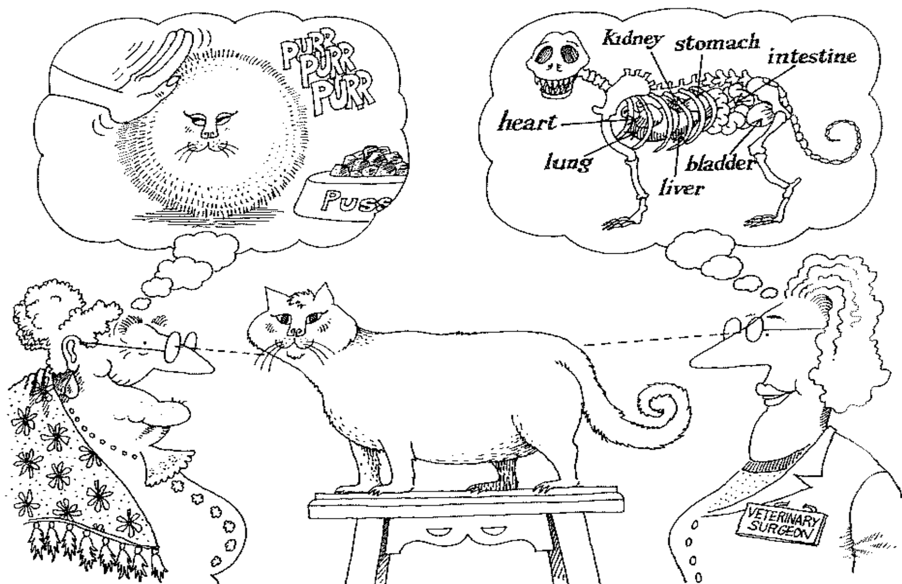
La gerarchia può essere di due tipi:

- Composizione (“ha un”, “parte di”)
- Ereditarietà (“è un”, “si comporta come”)

Il modello “a oggetti”

- Astrazione
- Incapsulamento
- Modularità
- Gerarchia
- Tipizzazione
- Concorrenza
- Persistenza

Astrazione/1



Astrazione/2

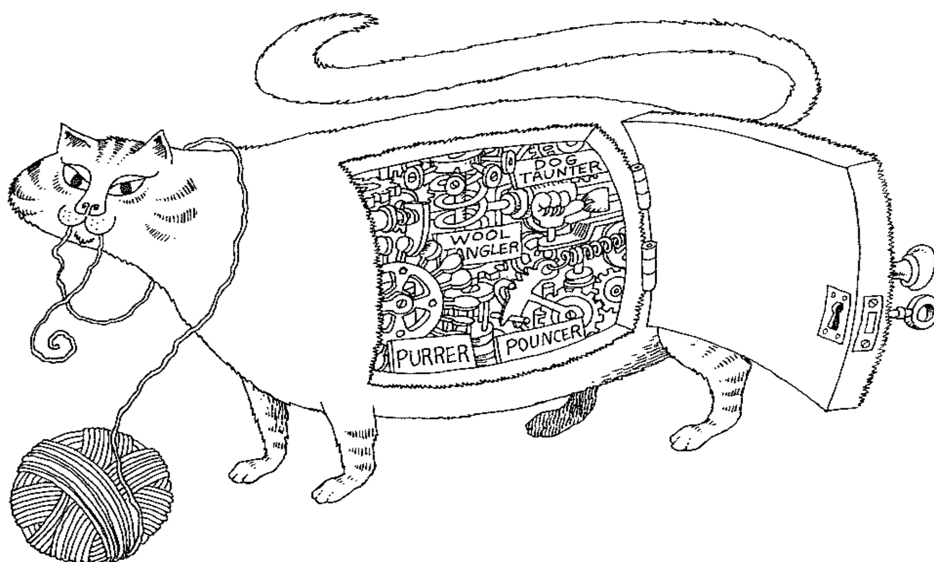
È la tecnica che consente di ridurre il contenuto informativo del problema da modellare, evidenziando solo gli aspetti interessanti.

Può riguardare sia i dati che le azioni eseguite sui dati; la OOP realizza entrambi gli aspetti.

Ad esempio: nel programma che gestisce il personale di un'azienda, una persona è caratterizzata da nome, codice fiscale, ruolo, dipartimento, anzianità, etc...

Per contro, nel programma che gestisce i pazienti di un ospedale, la stessa persona è caratterizzata da nome, età, altezza, peso, pressione, etc...

Incapsulamento/1



Incapsulamento/2

Lo scopo dell'incapsulamento consiste nel nascondere il comportamento interno degli oggetti.

Per usare una torcia elettrica basta sapere che si deve premere il pulsante; non occorre aprirla e vedere cosa c'è all'interno.

Analogamente, gli aspetti implementativi degli oggetti non devono essere accessibili agli utenti degli stessi.

Modularità/1

Modularità/2

La modularità consiste nello scomporre il sistema in componenti separate debolmente interdipendenti. Ciascuna componente è fortemente coesa.

In altre parole, i legami fra le parti interne di ciascun modulo sono più forti dei legami fra moduli.

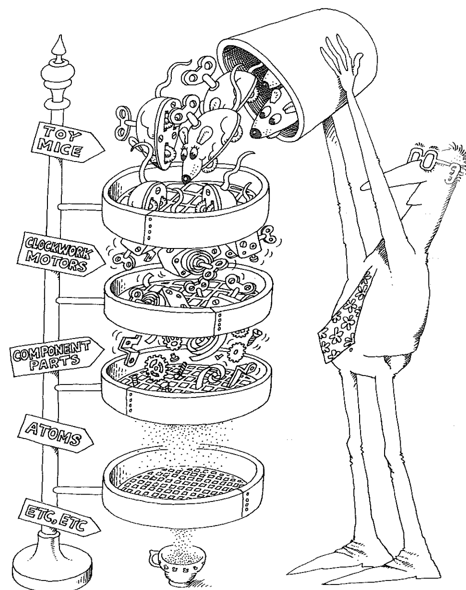
Modularità/3

Consideriamo ancora la torcia elettrica, e scomponiamola in sottosistemi:

- Lampada
- Pacco batterie
- Interruttore

Ciascuno è ben definito anche preso da solo. Inoltre avranno precise interfacce per le interdipendenze.

Gerarchia/1



Gerarchia/2

La gerarchia è semplicemente un ordinamento di astrazioni.

Spesso le astrazioni di concetti che entrano in gioco nel sistema hanno relazioni fra di loro; le relazioni sono di ereditarietà ("è un", "si comporta come") oppure aggregazione ("è parte di", "ha un").

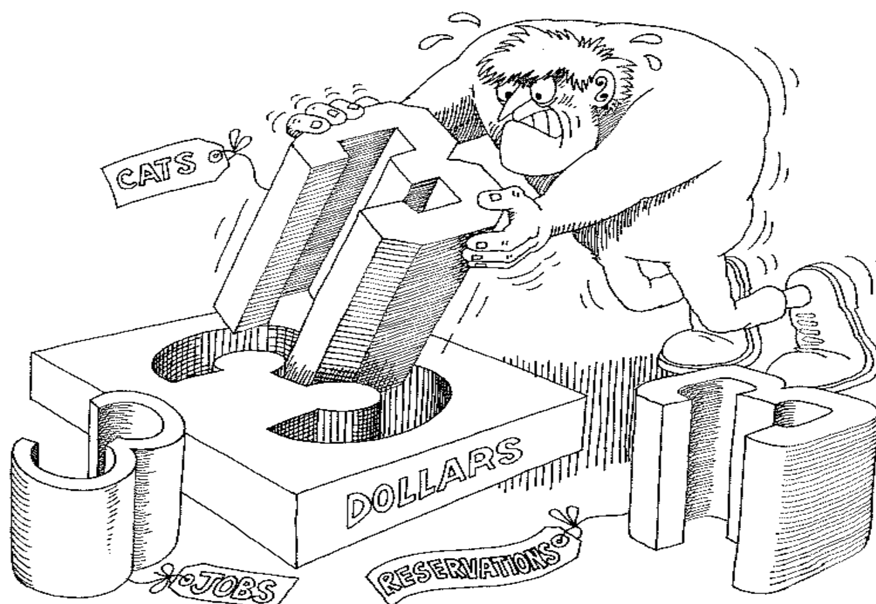
Gerarchia/3

Ad esempio, in un programma di simulazione di una automobile vi saranno varie entità: *ruota, motore, motore turbo, carrozzeria.* Una *automobile* è composta da tutte queste componenti (aggregazione).

Ma *motore* e *motore turbo* hanno molto in comune. Conviene progettare la macchina in modo che sia possibile montare l'uno o l'altro indifferentemente. Quindi è bene che abbiano la stessa interfaccia.

Inoltre un *motore turbo* è un *motore* con qualche pezzo in più (“è un” => ereditarietà).

Tipizzazione/1



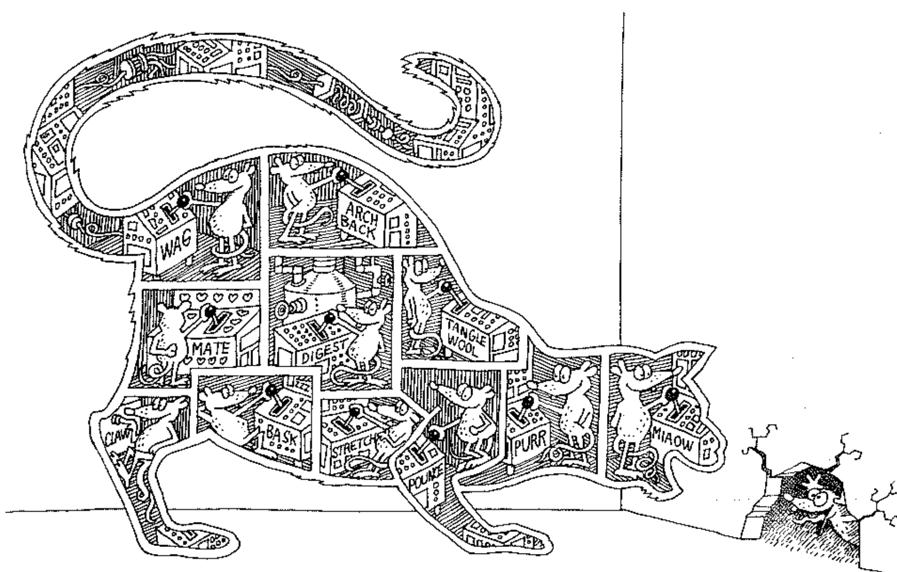
Tipizzazione/2

Un *tipo* è una caratterizzazione completa delle proprietà comportamentali e strutturali che sono in comune ad una classe di oggetti.

Il tipo è l'imposizione di una particolare classe ad un oggetto.

Oggetti di classi o tipi diversi non sono interscambiabili (oppure lo sono in modo molto ristretto)

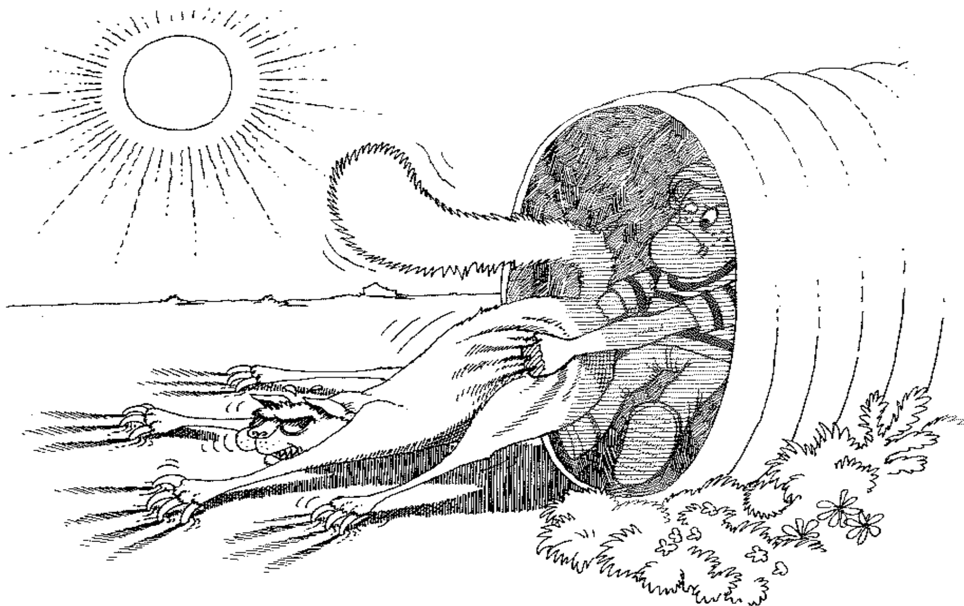
Concorrenza/1



Concorrenza/2

Spesso i sistemi complessi hanno più di una entità all'opera nello stesso istante di tempo (programmi paralleli o multithread).

Persistenza/1

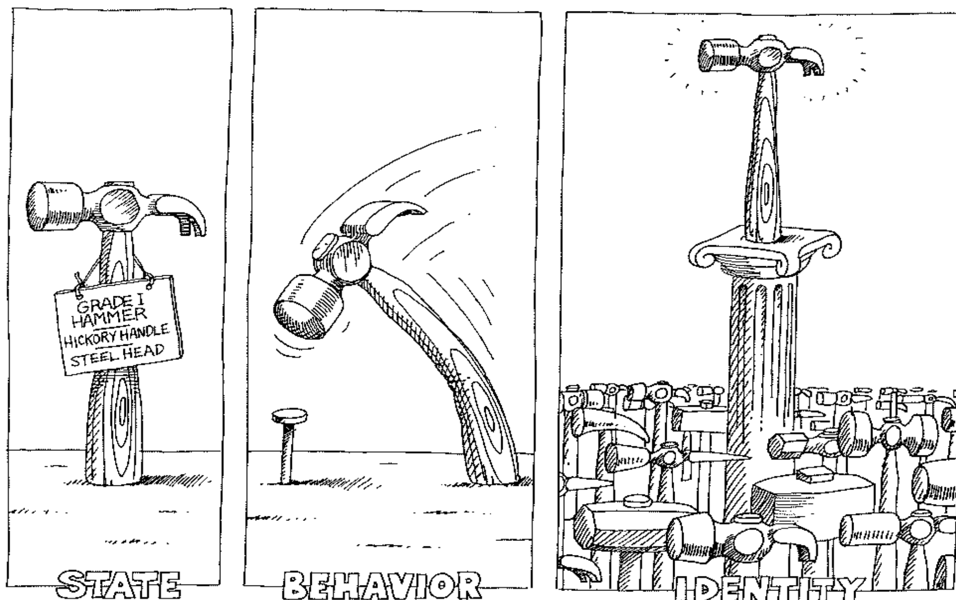


Persistenza/2

Si tratta della possibilità di realizzare oggetti persistenti rispetto al tempo (sopravvivono al programma che li ha creati) o allo spazio (possono essere migrati).

È un aspetto legato agli object-oriented database più che allo specifico linguaggio.

Oggetti/1



Oggetti/2

Gli oggetti sono entità concrete, caratterizzate da:

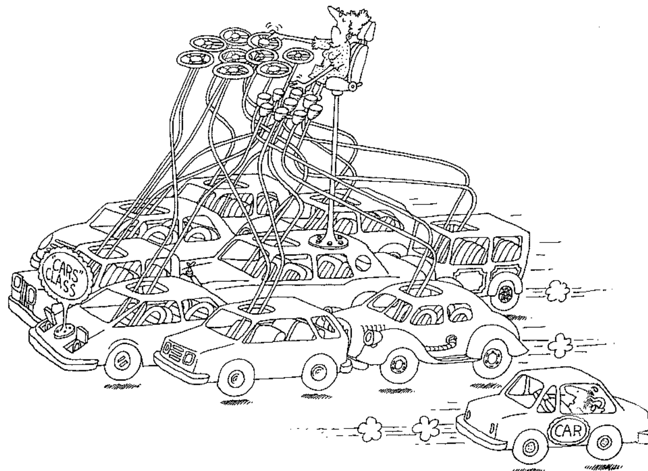
- STATO: le informazioni (proprietà) che caratterizzano l'oggetto
- COMPORTAMENTO: le azioni e reazioni che caratterizzano l'oggetto
- IDENTITÀ: l'oggetto è distinguibile da tutti gli altri oggetti

Struttura e comportamento di oggetti simili sono definiti nella classe comune ad essi.

Gli oggetti sono istanze di classi.

Classi

Una classe è un insieme di oggetti che condividono struttura e comportamento.



Ereditarietà

Una classe può ereditare la struttura ed il comportamento di una *superclasse*.

Classificazione/1

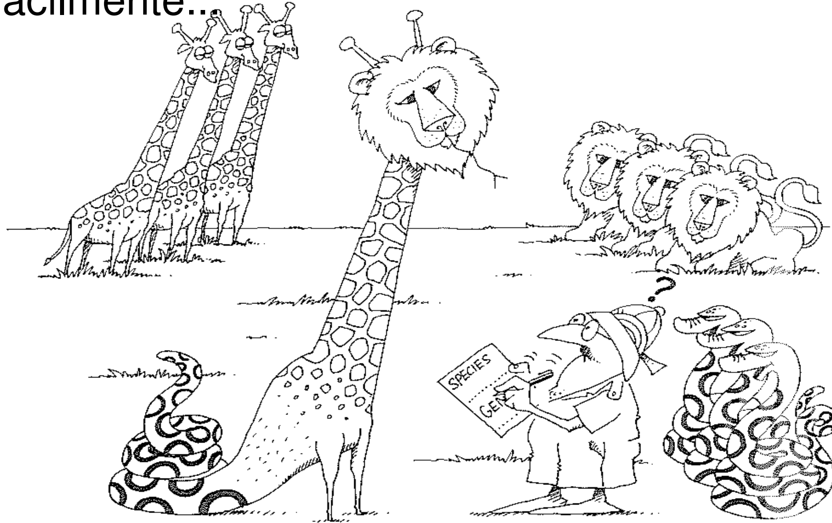
La classificazione è il modo naturale di ordinare la conoscenza.

In OOD la classificazione consiste nel riconoscere le giuste relazioni fra oggetti, e raggruppare gli oggetti in classi scelte in maniera adeguata.

Non è un processo facile!

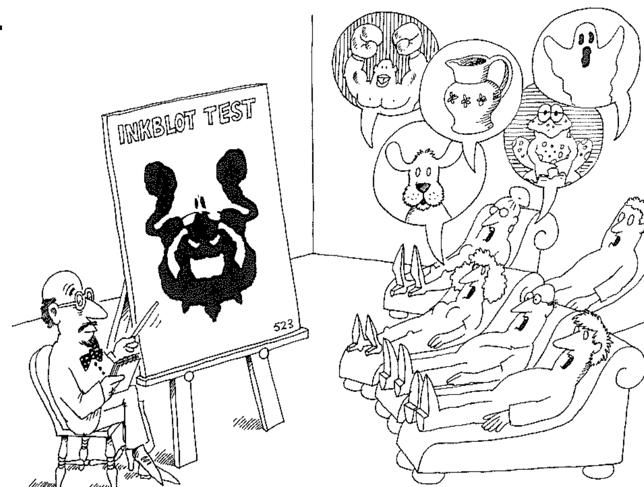
Classificazione/2

Certi oggetti non si lasciano classificare facilmente...



Classificazione/3

Osservatori differenti classificano in modo differente...



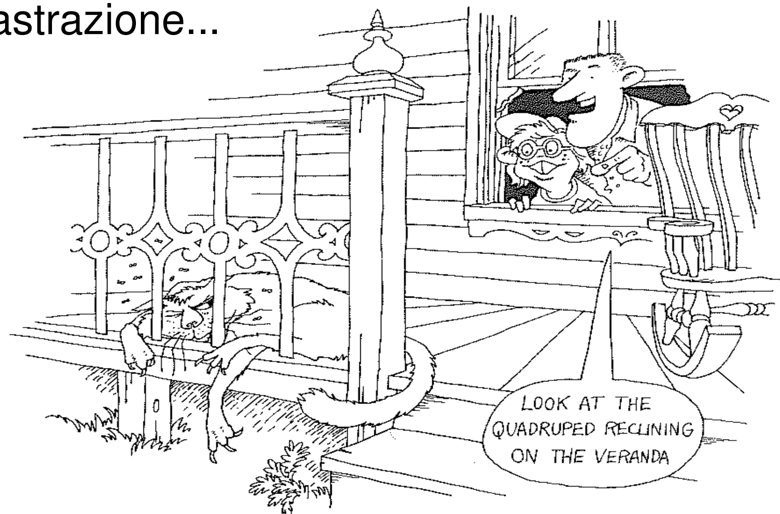
Astrazioni chiave/1

Una astrazione chiave è una classe o oggetto che fa parte del vocabolario del problema.

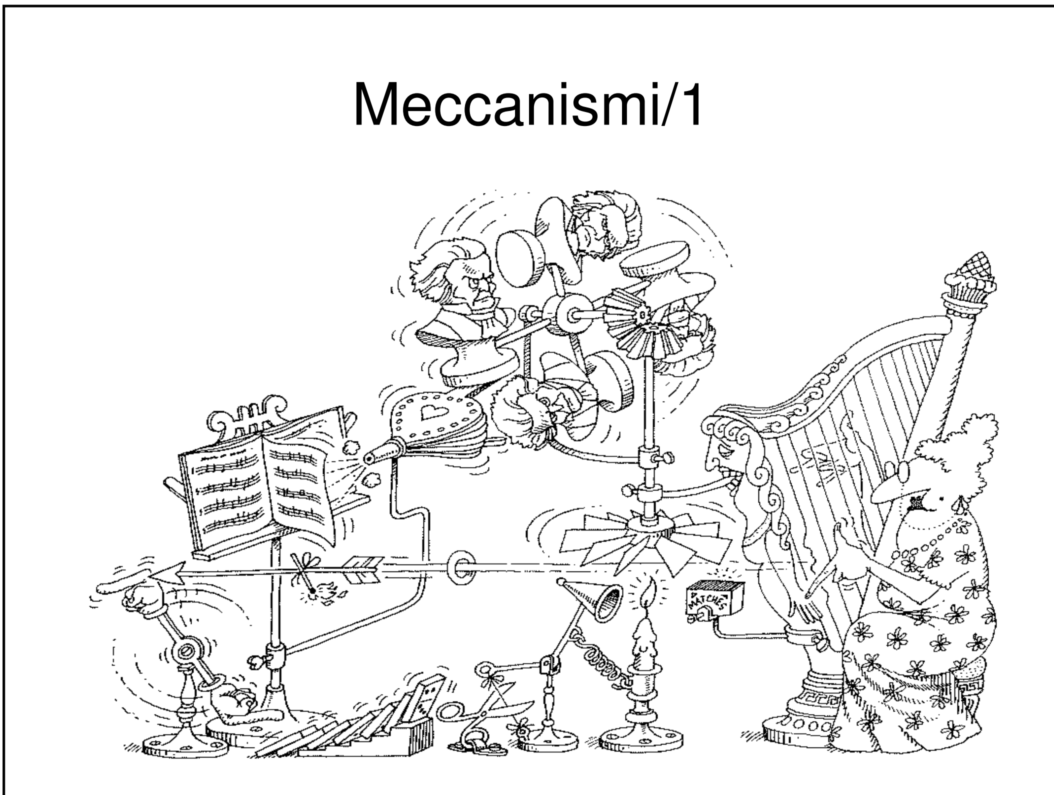
Ad esempio: Una automobile è composta da un motore, quattro ruote, una carrozzeria. Il motore turbo è un particolare tipo di motore.

Astrazioni chiave/2

È importante scegliere il giusto livello di astrazione...



Meccanismi/1



Meccanismi/2

I meccanismi sono la maniera con cui vari oggetti collaborano per realizzare un comportamento di alto livello.

Ad esempio: Il motore mette in rotazione l'albero motore, che assieme alla scatola del cambio trasferisce il moto all'albero di trasmissione, il quale a sua volta attraverso il differenziale lo trasmette alle ruote.

Esempio

Il sistema da rappresentare è composto da figure geometriche.

Quadrati, rettangoli, cerchi e triangoli sono figure geometriche.

Una figura geometrica ha una posizione ed un colore; alcune figure geometriche hanno altri parametri: il quadrato un lato, il rettangolo due, il triangolo tre ed il cerchio ha un raggio.

Una figura geometrica può essere disegnata sullo schermo, cancellata, spostata (cambio di posizione).

Programmazione procedurale

- Definisco le variabili necessarie per rappresentare i dati
- Definisco funzioni *sposta_quadrato*, *sposta_cerchio*, *disegna_triangolo* etc... che non hanno alcuna relazione esplicita fra di loro, e prendono vari parametri
- I legami fra dati ed azioni non è esplicito

OOP/1

- Si individuano gli oggetti che fanno parte del sistema
- Si individuano gli attributi o proprietà degli oggetti (dati)
- Si individuano i metodi (comportamento) degli oggetti (funzioni)
- Si classificano gli oggetti individuando le giuste astrazioni
- Gli oggetti sono raccolti in classi

(quadrati, cerchi, ...)

- Possono esservi gerarchie fra classi

(quadrato → FormaGeometrica)

- Per una data classe di oggetti, attributi e metodi sono raccolti assieme

Stato - Attributi

Un oggetto è composto da attributi:

```
Quadrato q;
q.lato = 10;
```

Comportamento - Metodi

La conoscenza di come un oggetto agisce o reagisce è contenuta nello stesso oggetto, non all'esterno. Quindi, non si fa così:

```
disegna_quadrato(q);
```

ma piuttosto così:

```
q.disegna();
```

Accesso

Gli oggetti nascondono vari aspetti di implementazione; normalmente non si dà accesso agli attributi:

```
q.lato = 3;   //SYNTAX ERROR!
```

Costruzione

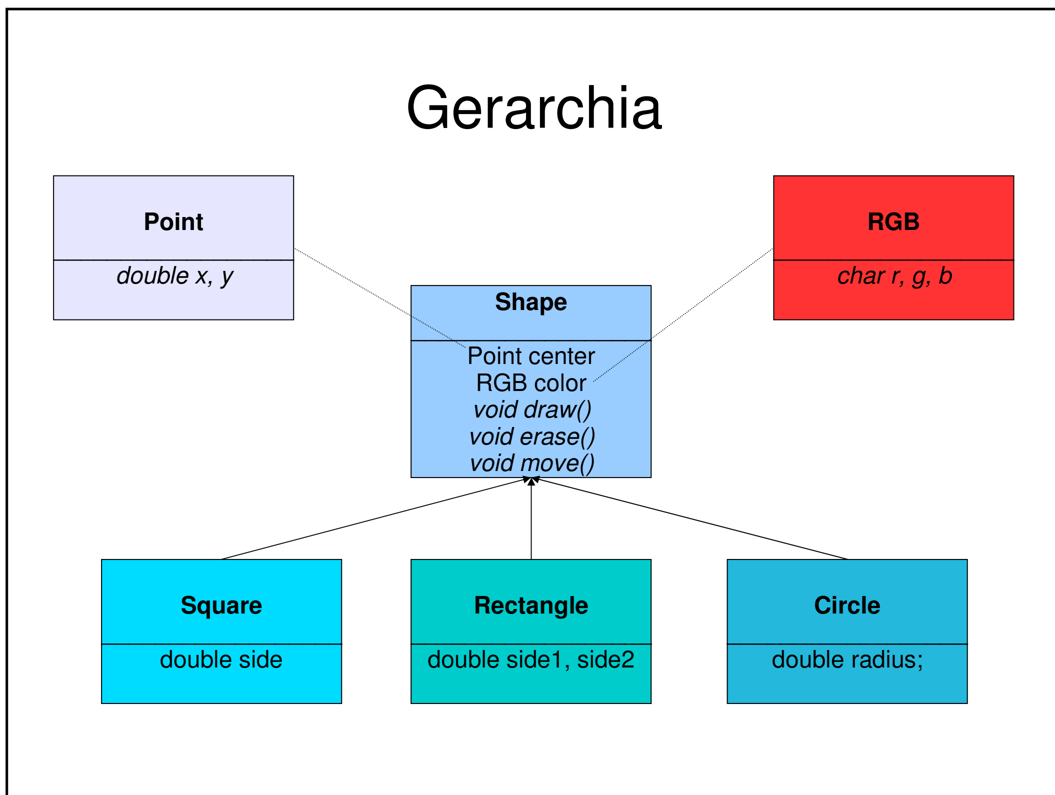
Occorre specificare come costruire un oggetto:

```
Quadrato q(
    Punto(0.0,0.0),        //posizione
    Colore(255,255,255),   //colore
    1.5                    //lato
);
```

OOA/OOD

Da dove partire con OOA/OOD? Attenzione alla descrizione del problema in linguaggio naturale!

- Sostantivi => classi o oggetti (figura geometrica, quadrato, cerchio, rettangolo, triangolo, posizione, colore, ...)
- Verbi/azioni => metodi (disegnare, cancellare, spostare)
- Relazioni:
  - Ha un => aggregazione (Una forma geometrica **ha una** posizione)
  - È un => ereditarietà (Un quadrato **è una** forma geometrica)



Classe Shape

```
class Shape {
  protected:
    Point center; RGB color; bool painted;
  public:
    Shape(Point p, RGB c) {
      center = p; color = c; painted = false; }
    void move(Point p) {
      erase(); center = p; draw(); }
    virtual void erase() =0;
    virtual void draw() =0;
};
```

Classe Square

```
class Square : public Shape {
    double side;
  public:
    Square(Point p, RGB c, double s) : Shape(p,c) {
      side = s; }
/*  void move(Point p) { NO NEED TO DEFINE THIS AGAIN!
      erase(); center = p; draw(); } */
    virtual void erase() {
      /* interesting code is here! */ }
    virtual void draw() {
      /* interesting code is here! */ }
};
```

Classe Rectangle

```
class Rectangle : public Shape {
    double side1, side 2;
  public:
    Rectangle(Point p, RGB c, double s1, double s2) :
                Shape(p,c) {
      side1 = s1; side2 = s2 }
    virtual void erase() {
      /* interesting code is here! */ }
    virtual void draw() {
      /* interesting code is here! */ }
};
```

Uso della libreria

```
int main() {
  Square    s( Point(0.5,-1.3),
               RGB(100,255,255),
               0.98 );
  Rectangle r( Point(1.0,0.0),
               RGB(255,255,0),
               3.21, 4.45 );
  s.draw();
  r.draw();
  s.move( Point(1.1,2.1) );
}
```

Polimorfismo/1

Il polimorfismo consiste nella possibilità di utilizzare in maniera omogenea oggetti di tipo diverso. Occorre ovviamente che abbiano la stessa interfaccia; ciò è garantito dall'ereditarietà.

Grazie al polimorfismo, si possono scrivere funzioni che operano su qualsiasi forma geometrica.

Polimorfismo/3

```
void draw_all_shapes(vector<Shape*> v) {
  for(vector<Shape*>::const_iterator vi=v.begin);
      vi!=v.end(); ++vi) {
    Shape * s = *vi;
    s->draw();
  }
}
```

Polimorfismo/4

```
int main() {
  Square s(...);
  Rectangle r(...);
  vector<Shape *> v;
  v.push_back( &r );
  v.push_back( &s );
  v.push_back( read_shape_from_file("some.shp") );
  draw_all_shapes(v);
}
```

Tecniche per una buona programmazione OOP

- Scrivere sempre prima il programma di test!
- Nell'ordine:
  - Individuare gli oggetti
  - Definire l'interfaccia degli oggetti
  - Definire la rappresentazione degli oggetti
- UML
- Design patterns