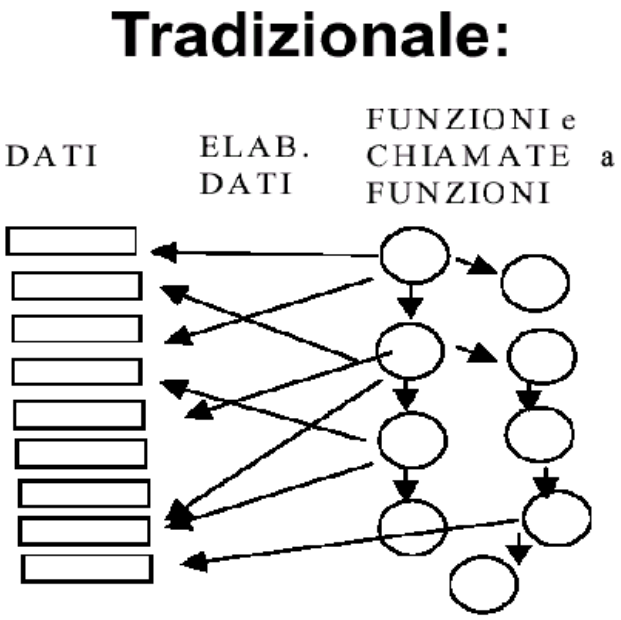
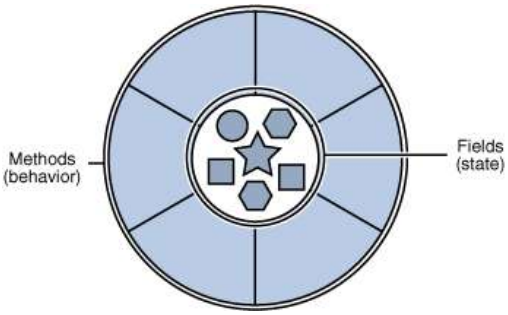
La programmazione: tradizionale vs Orientata agli Oggetti (OOP)

In generale, nella soluzione di un problema si individuano dati e funzionalità (azioni) potendola implementare, a livello più semplice, in un programma costituito da *istruzioni* e dati:

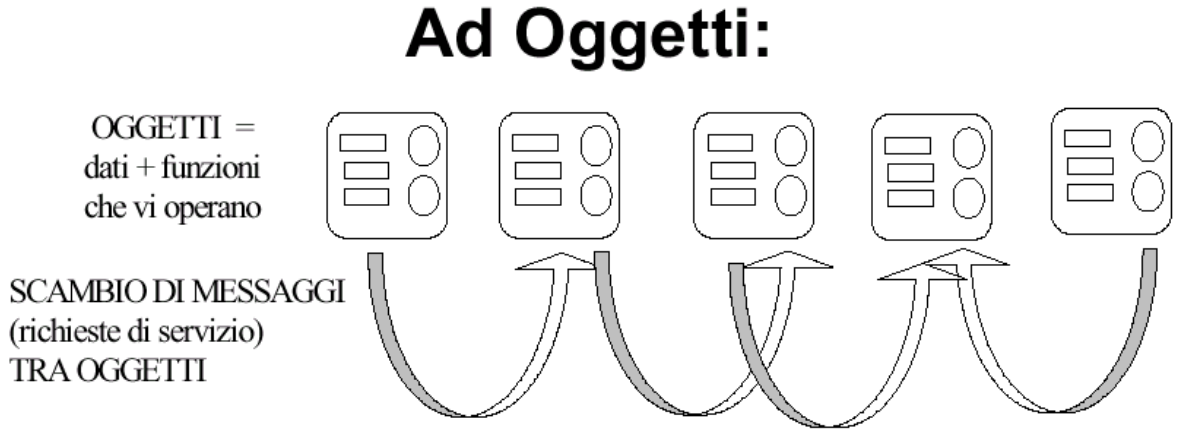


nella progettazione **OO** il codice che manipola i dati è invece ***incapsulato*** con la dichiarazione e memorizzazione di quei dati.



Si pensi all'incapsulamento come ad un involucro protettivo che avvolge sia le *istruzioni* che i dati che si stanno manipolando. Tale involucro definisce il comportamento e protegge da accessi arbitrari da parte di un altro programma; il pregio è che ognuno può accedere al codice e usarlo senza preoccuparsi dei dettagli di implementazione.

L'essenza della **programmazione Orientata agli Oggetti** è trattare come entità concrete *oggetti* anche astratti con un proprio specifico comportamento che *rispondono a messaggi* che dicono loro di fare qualcosa; una sequenza di passi di un procedimento tipica della programmazione con paradigma "procedurale" può diventare una collezione di messaggi tra oggetti autonomi.



Quindi, all'interno dell'impostazione procedurale, si introduce un ***GROSSO CAMBIAMENTO DI PARADIGMA:***

NON si richiede a funzioni di operare su dati
nome_funzione (parametro1, parametro2, ….);
MA si richiede alle entità di eseguire delle funzioni
oggetto.nome_funzione (parametro1, parametro2, ….);

MODELLO CLIENTE/SERVITORE, quindi, con spostamento del "fuoco"

NON chiamare una funzione che elabori dei dati
MA chiedere a una entità software di svolgere un servizio

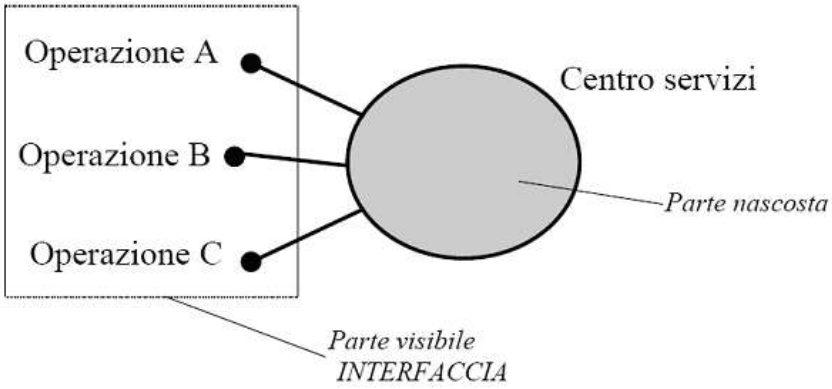
*Esempi*o: dato un flusso, NON invocare una funzione che scriva dati sul flusso,
MA chiedere al ***flusso*** stesso di svolgere il servizio relativo alla **scrittura**

***System.out*.println(*stringa*)**

- I clienti *non conoscono* l'organizzazione interna dei centri di servizio, e *non possono* accedere *direttamente* a essa
- non interessa l'algoritmo (come sono fatte le cose) ma interessa che le *cose vengano fatte*
- i clienti sono indipendenti da come sono fatti i servitori: facilita la manutenzione!

IL CONCETTO DI OGGETTO

- Un oggetto è inteso come un ***centro di servizi***, un'***astrazione di dato***
  - con una *parte visibile* → ***INTERFACCIA***
  - ed una *parte nascosta*



LE OPERAZIONI DI *INTERFACCIA* RAPPRESENTANO:

- I servizi che gli altri oggetti possono richiedere all'oggetto

- Modalità di trattamento dei dati dell'oggetto o di accesso ai dati

LA *PARTE NASCOSTA* CONTIENE:

DATI che sono accessibili solo attraverso richieste di servizio:
- Gli attributi che caratterizzano l'oggetto
- Lo stato dell'oggetto

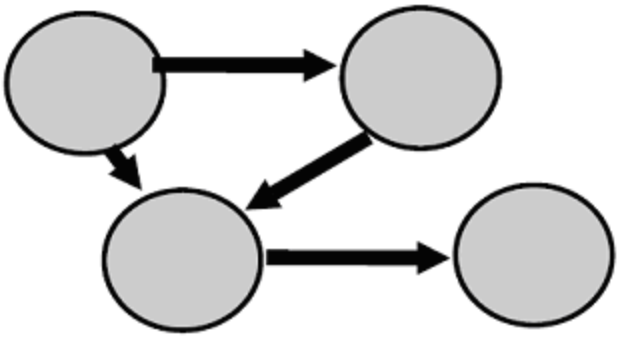
OPERAZIONI PRIVATE, sfruttate dalle operazioni pubbliche ma non servizi esterni

DESCRIZIONE DI UN SISTEMA A OGGETTI

- Ogni oggetto appartiene a ("è istanza di") una data ***classe***
- **La classe *racchiude e incapsula la specifica*** di:
    - **struttura** dell'oggetto (dati)
    - **comportamento** dell'oggetto (operazioni)
- Le classi possono essere correlate tra loro (*tassonomie* di ***ereditarietà***) in modo da permettere il collegamento tra classi simili per raggruppare le proprietà simili e, in ultima analisi, per ***riusare facilmente*** il codice comune alle classi correlate e ridefinire velocemente nuove classi

ARCHITETTURA DI UN SISTEMA A OGGETTI

- Un ***insieme di oggetti***
    - che **interagiscono gli uni con gli altri**
    - senza conoscere nulla delle rispettive rappresentazioni concrete



VANTAGGI[1] DELLA PROGRAMMAZIONE AD OGGETTI

Facilitazione di costruzione *cooperativa* di software:

- diverse persone sviluppano diverse classi
- ogni programmatore può semplicemente verificare il comportamento delle sue classi istanziandone oggetti e verificandone il comportamento in risposta a richieste di servizio
- unico accordo necessario per integrare il tutto in un unico sistema finale: definire le interfacce delle classi

Facilitazione della *gestione e manutenzione*

- Se vi sono errori sui dati in un oggetto, è facile scoprire dove si trova l'errore, perché (siccome i dati non sono visibili all'esterno dell'oggetto) esso non potrà essere che all'interno dell'oggetto che gestisce quei dati
- le modifiche a una classe non rendono necessario modificare il resto del programma (le altre classi) a meno che non venga modificata l'interfaccia

Supporto a progettazione e sviluppo *incrementali* : si possono definire nuove classi sfruttando il codice di classi già esistenti

Possibilità di *rapida prototipazione* : non importa che una classe sia completamente definita per poter iniziare a "testare" il funzionamento degli oggetti di quella classe.

[1] Interessante lettura "Linguaggi e Metodologie Orientate agli Oggetti" (*Object Oriented Analysis* ed *Object Oriented Design*)

Principi su cui si basa un linguaggio ad oggetti

Ecco quindi quelli che generalmente sono considerati i principi su cui si basa un **linguaggio ad oggetti**:

1. "Ogni cosa è un oggetto". Per comprendere meglio tale affermazione si può pensare all'oggetto come ad una *particolare variabile*. Particolare perchè riceve *dati* ma può anche compiere *operazioni* su sé stessa.

Un generico albero può essere caratterizzato dall'**altezza** (`height` variabile nel tempo)

e dal ***metodo*** che determina la sua crescita ad esempio di un metro

2. "Un programma è una serie di oggetti in comunicazione tra di loro". La comunicazione tra gli oggetti avviene principalmente attraverso *messaggi*. Scendendo più nel dettaglio, si può pensare ad un messaggio come ad una *richiesta* fatta all'oggetto di richiamare una funzione di competenza dello stesso.

3. "Ciò che risulta dall'insieme di più oggetti è esso stesso un oggetto". La questione riflette in maniera abbastanza esplicita il riferimento al punto 1: se "ogni cosa è un oggetto" anche tanti oggetti messi insieme sono un oggetto.

4. "Ogni oggetto appartiene a un tipo definito" ossia ogni oggetto è un'istanza di una classe.

   Per adesso basterà sapere che una ***classe*** è il tipo (o la categoria) di oggetto, ad esempio la categoria delle auto, delle moto, delle navi e così via.

   L'***istanza*** è invece un particolare elemento della categoria. Prendiamo l'esempio delle automobili: la classe rappresenta il concetto astratto di "automobile". L'istanza è per esempio la propria macchina,

   quella del vicino di casa,

   e così via.

5. "Tutti gli oggetti facenti parte della stessa categoria possono ricevere i medesimi messaggi" questo significa che entrando in due macchine diverse si può essere sicuri di avere la possibilità di accendere i fari perché entrambe le automobili supportano questo tipo di **messaggio 'accendi i fari'**. In generale quindi se si ha a che fare con un elemento dell'insieme 'automobili' si può compiere ogni operazione consentita dalla categoria.

6. "Ogni oggetto è dotato di un'interfaccia" oggetti del tutto identici tra loro tranne che per lo stato che possono assumere (o che assumono) nell'esecuzione di un'applicazione (si pensi ad esempio a due lampadine, una accesa ed una spenta) sono raggruppabili come già detto in classi di oggetti.

   Una volta che una classe è definita, è possibile creare tanti oggetti (ossia tanti elementi della classe) quanti ne servono per lo sviluppo dell'applicazione.

Ma come si può manipolare uno di questi oggetti? Ogni volta che chiediamo ad un oggetto di compiere una determinata operazione, interagiamo con l'interfaccia dell'oggetto, differente in base alla classe alla quale esso appartiene. Si pensi ad esempio ad una lampadina, come nella figura sottostante:

La categoria delle lampadine, ossia la classe, è chiamata 'Lampadine'. La particolare lampadina che prendiamo in considerazione è quella del salotto di casa, e ciò che può essere fatto con essa (in parole povere il suo *scopo*) è *definito dall'interfaccia*. La lampadina può allora essere accesa, spenta o diffondere una luce soffusa.

Ad esempio, riprendendo l'esempio dell'albero, in linguaggio Java si creerà una classe di nome `Tree` con un attributo (`height`) nascosto, inizialmente nello *stato* `height=0` e come interfaccia, il metodo che modifica tale dato incrementandolo di uno:

```
class Tree {
                    /**
                        altezza dell'albero
                    */
                    private int height;     // attributo nascosto
                                            // inizializzato di default a zero
                    /**
                        Crescita di un metro
                    */
                    public void cresci() {

                              height = height + 1;
                    }

                    public static void main(String [] args) {

                              // Creazione di un oggetto della classe Tree

                              Tree tree1 = new Tree();

                              // richiesta di crescere di un metro

                              tree1.cresci();   // accesso al metodo
                                                // con l'operatore dot
                    }
          }
```

Nb:

Sintassi per creare un'**istanza** di una classe (cioè un oggetto memorizzato in RAM):

```
NomeClasse  nomeOggetto =  new NomeClasse();   // metodo costruttore
                                               // con lo stesso nome della classe
```

Sintassi per chiedere, come *servizio*, ad un oggetto di eseguire un suo metodo:

```
nomeOggetto.nomeMetodo();   // operatore dot per accedere al metodo
                            // chiedendo all'oggetto di eseguire quell'azione
```