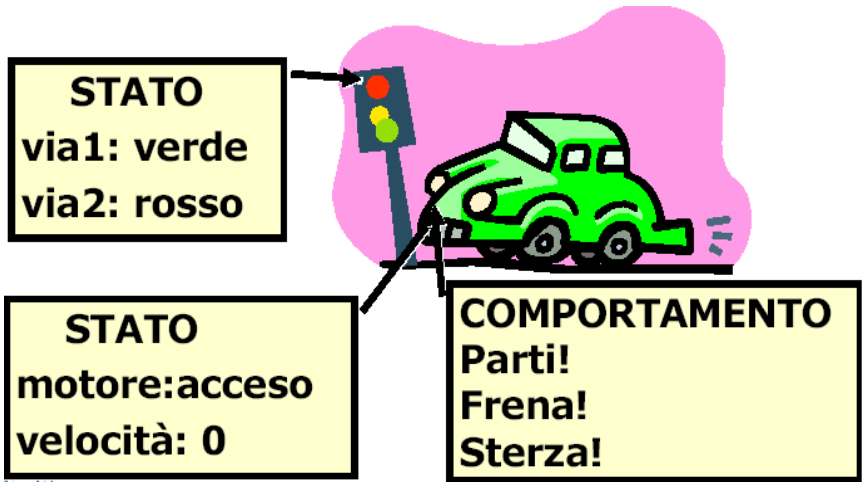
Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica

Programmazione I - corso B
a.a. 2009-10

prof. Viviana Bono

Blocco 13 – Introduzione alla progettazione a oggetti
(autori vari)

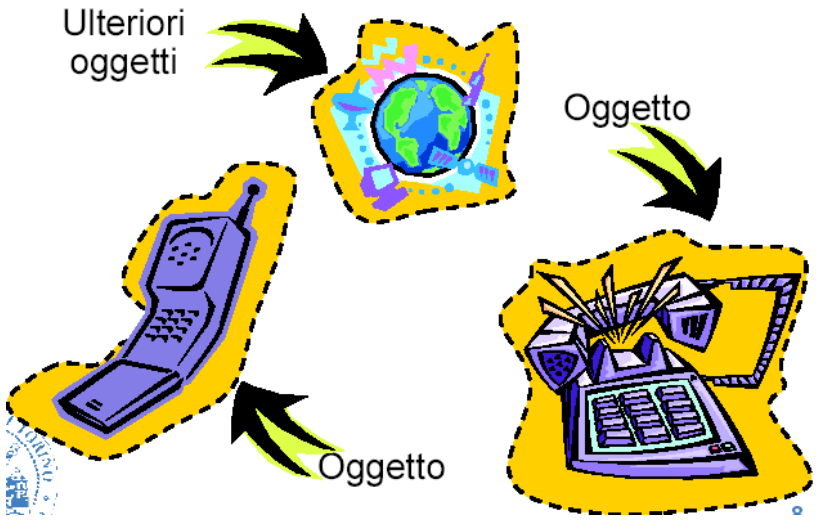
Modellare la realtà I



Modellare la realtà II

- Stato
  - L'insieme dei parametri caratteristici che contraddistinguono un oggetto in un dato istante
  - Modellato come insieme di attributi
- Comportamento
  - Descrive come si modifica lo stato a fronte degli stimoli provenienti dal mondo esterno
  - Modellato come insieme di metodi

Approccio nell'osservare il mondo



Oggetti e realtà

- Il mondo fisico è costituito da un insieme di oggetti variamente strutturati che interagiscono tra loro
- Ciascuno è dotato di:
  - Una propria **identità** (è riconoscibile)
  - Uno **stato** (ricorda la storia passata)
  - Un **comportamento** (reagisce a stimoli esterni in un modo prevedibile)
- Si può estendere la metafora al software
  - Ogni entità logica che deve essere manipolata può essere immaginata come un "oggetto"

Stato

- Ogni oggetto ha uno stato:
  - L'insieme dei parametri caratteristici che contraddistinguono un oggetto in un dato istante
  - Riflette la storia dell'oggetto
- Composto da un un gruppo di "attributi"
  - Ogni attributo modella un particolare aspetto dello stato
  - Può essere un valore elementare o un altro oggetto…
- Implementato mediante un blocco di memoria
  - Contiene i valori degli attributi
- Principio fondamentale: **incapsulamento**
  - Lo stato "appartiene" all'oggetto
  - Un utente esterno non può manipolare direttamente lo stato

Comportamento

- Gli oggetti interagiscono a seguito di "richieste esterne"(messaggi)
  - Dotate di eventuali parametri che ne specificano i dettagli
- Ogni oggetto sa reagire ad un ben determinato insieme di messaggi
  - Costituiscono la sua interfaccia
- Ad ogni richiesta è associato un comportamento
  - Modifica dello stato
  - Invio di richieste verso altri oggetti
  - Comunicazione di informazioni (risultato)
- Implementato attraverso un blocco di codice (metodo)
- Contiene la sequenza delle operazioni da svolgere
- Principio fondamentale: **delega**
- Chi effettua la richiesta non vuole conoscere i dettagli di come la richiesta sia evasa

La programmazione orientata agli oggetti (secondo Alan Kay – Smalltalk)

- Ogni cosa è un *oggetto*
- Un programma è un insieme di *oggetti* che si "dicono l'un l'altro" che cosa fare "inviandosi" *messaggi*
- Ogni *oggetto* può contenere riferimenti ad altri *oggetti*
- Ogni *oggetto* ha un tipo (*classe*), cioè ogni oggetto ha proprietà strutturate in campi definiti dalla *classe*
- Tutti gli *oggetti* di un determinato tipo possono ricevere gli stessi *messaggi*

Tipo di dato astratto

- Definisce le operazioni fondamentali sui dati, ma non ne specifica l'implementazione

  Per es. una *lista* (astratta) è una sequenza ordinata di dati

- le operazioni possibili sono:
  - lettura sequenziale
  - inserimento/rimozione di un elemento in posizione i-esima

  Una **struttura-dati** è vista come l'insieme di operazioni (*servizi*) offerti

Il ruolo dell'astrazione

- Astrazioni procedurali
- Astrazioni dei dati

Obiettivo: trattare cose complesse come primitive e nascondere i dettagli

Domande:

- **Quanto è facile suddividere il sistema in moduli di astrazioni?**
- **Quanto è facile estendere il sistema?**

La programmazione orientata agli oggetti

Object-oriented design = progettazione (e sviluppo) orientato agli oggetti = costruzione di sistemi software visti come collezioni strutturate di (implementazioni di) strutture-dati astratte [B. Meyer, "Object-oriented software construction ", Prentice Hall, 1988, cap.4.8]

Programmazione: procedurale vs OO

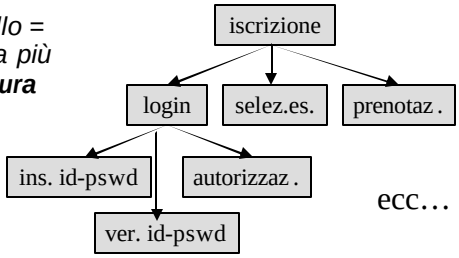
- Programmazione procedurale
  - Organizzare il sistema intorno alle procedure che operano sui dati
- Programmazione ad oggetti
  - Organizzare il sistema intorno ad oggetti che si scambiano i messaggi
  - Un oggetto *incapsula* dati e operazioni

La programmazione procedurale

Metodo classico di software design = *top-down functional (structured) design* = scomposizione gerarchica funzionale (algoritmica)

paradigma procedurale: algoritmo = procedura = sequenza di passi per raggiungere il risultato

*Per es. iscrizione ad un appello = scomposizione in passi via via più semplici, elementari = **procedura** per iscriversi ad un appello*



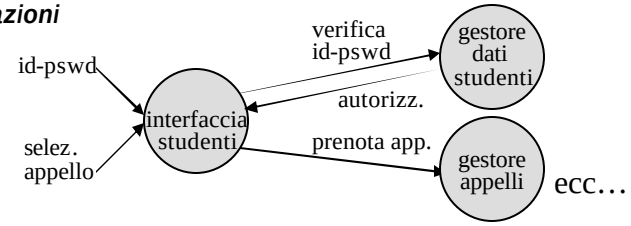
PROGRAMMI = ALGORITMI + STRUTTURE-DATI

La programmazione orientata agli oggetti

Metodo alternativo = ***object-oriented design*** = si parte dagli oggetti, non dalle funzionalità!

paradigma ad oggetti: oggetti che interagiscono tra loro scambiandosi dei messaggi = collaborazione per raggiungere il risultato

*Per es. iscrizione ad un appello = **entità** coinvolte nell'attività e loro **relazioni***



PROGRAMMI = OGGETTI (DATI + ALGORITMI) + COLLABORAZIONE (INTERFACCE)

Sviluppare un programma ad oggetti

- Un oggetto è un *fornitore di servizi*
- Un programma fornisce un servizio agli utenti e lo realizza utilizzando servizi di altri oggetti

- Obiettivo: produrre (o trovare librerie di oggetti già esistenti) l'insieme di oggetti che forniscono i servizi ideali per risolvere il problema

Progettare ad oggetti

- Si parte dal testo delle specifiche
  - Si individuano i nomi e i verbi
- Tra i nomi, si individuano le possibili classi di oggetti
  - Con i relativi attributi
- Tra i verbi, si individuano metodi e relazioni
  - Di solito, i verbi di azione si modellano come metodi ("X apre Y"), quelli di stato come relazioni ("A si trova presso B")
  - L'interazione tra due oggetti sottende l'esistenza di una relazione tra gli stessi
  - Attenzione alle forme passive e ai sostantivi deverbali!

Processo di creazione del software

- E' utile definire come **verificare**
  - Progetto del test: definisce i controlli da attuare
  - Traduce i requisiti in un insieme di misure
- Da ultimo, si **scrive** e **verifica** il codice
  - Sviluppo: produce la soluzione
  - Implementa l'architettura
- Il processo è **iterativo**
  - Le diverse fasi si susseguono raffinando progressivamente la soluzione

Oggetti

- Servono a modellizzare oggetti che appartengono al dominio del problema.
- Esempio: se vogliamo simulare il traffico stradale, una delle entità è l'*automobile*.
  - Automobile è classe o oggetto ?
  - Per capirlo rispondiamo alle domande:

    1. Di che colore è?
    2. A quale velocità arriva?
    3. Dove si trova adesso?

La programmazione orientata agli oggetti

Che cos'è un "oggetto"?

Passo 1: Distinguere classi e istanze

Passo 2: Distinguere interfaccia e implementazione

Passo 1: Distinguere classi e istanze

- Un'**istanza** (**oggetto**) è un'entità concreta, che esiste nel tempo (viene costruita e poi distrutta) e nello spazio (occupa memoria)
- Una **classe** è un'astrazione che rappresenta le proprietà comuni (struttura e comportamento) ad un insieme di oggetti concreti (istanze)

La programmazione orientata agli oggetti

Esempio: supponiamo di gestire una biblioteca, che contiene moltissimi libri

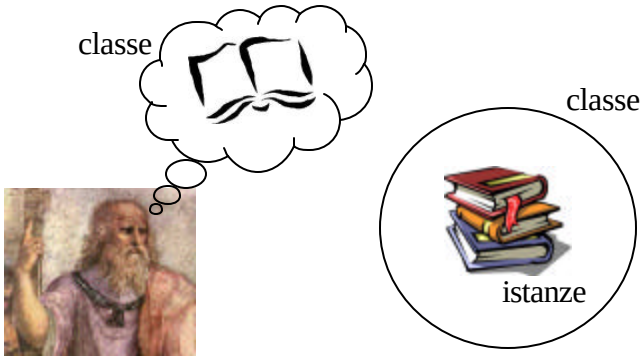
Una classe è…

- L'insieme di tutti i libri, la classe dei libri
- La proprietà *Libro(x)*, che definisce l'appartenenza all'insieme dei libri, ed è vera per tutti i libri (gli oggetti *x* che sono libri)
- L' "idea platonica" di libro, il prototipo ideale di libro, che esiste solo nel mondo delle idee; tutti i libri della nostra biblioteca "partecipano" dell'idea di libro, da momento che sono libri!
- Il concetto mentale di libro, che esiste solo nella nostra testa e di cui i libri del mondo sono degli esempi concreti
- …

La programmazione orientata agli oggetti

Un'istanza (oggetto) è…

- Un singolo libro concreto (che può essere preso in prestito, restituito, distrutto, fotocopiato, ecc...)



La programmazione orientata agli oggetti

Þ Una **classe** può essere vista come la definizione di un **tipo** (astratto)

Per esempio, supponiamo che la biblioteca riceva un nuovo libro. Per prima cosa il bibliotecario deve classificarlo come libro (e non rivista, CD Rom, o altro), dichiarando quindi che l'oggetto appena arrivato è un libro.
Questo equivale a dichiarare che **il nuovo oggetto è di tipo "libro"** (cioè che alla domanda "cos'è questo?" rispondiamo "è un libro!")

Un tipo è un **modello** (un ***template***) che definisce il **comportamento** (e la **struttura**) di un'insieme di istanze (oggetti).

Per esempio, il tipo "libro" definisce le operazioni che possono essere fatte sui libri (istanze): prestito, restituzione, ecc...

La programmazione orientata agli oggetti

Þ Un'**istanza** è un oggetto concreto (di un certo tipo, cioè appartenente ad una certa classe), caratterizzato da:

- un'**identità**: possibilità di identificare univocamente l'oggetto
- uno **stato**: l'insieme dei valori dei suoi attributi, in un certo tempo *t*
- un **comportamento**: l'insieme delle operazioni (funzionalità) offerte dall'oggetto, cioè le cose che l'oggetto è in grado di fare

La programmazione orientata agli oggetti

Torniamo al nostro esempio:

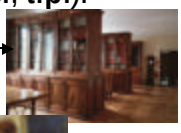
supponiamo di gestire una biblioteca, che contiene molti libri; nella biblioteca c'è un bibliotecario che classifica i nuovi libri, assegna i prestiti, ecc. e ci sono degli utenti che prendono in prestito i libri della biblioteca

Quali sono gli **oggetti** coinvolti nello scenario?

In particolare, quali sono le **classi** e quali le **istanze**?

Abbiamo bisogno dei seguenti **concetti** (**classi**, **tipi**):

- la Biblioteca
- i Bibliotecari
- il Libro
- l'Utente




La programmazione orientata agli oggetti

Per ogni **concetto** (**classe**, **tipo**) di quali **proprietà** (**attributi, caratteristiche**) abbiamo bisogno per descriverlo in modo adeguato?

- per la Biblioteca:
  - nome
  - indirizzo
  - orario apertura

- per il Bibliotecario:
  - nome
  - turno

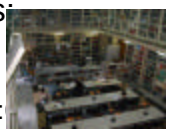
- per il Libro:
  - autore
  - titolo
  - editore
  - collocazione

- per l'Utente
  - nome
  - cognome
  - telefono

La programmazione orientata agli oggetti

Per ogni concetto (classe, tipo) di quante **istanze** (oggetti concreti) abbiamo bisogno?

- **una sola biblioteca** (un'istanza della classe Biblioteca), per la quale, per es:
  - nome = Biblioteca A. Gramsci
  - indirizzo = via Tizio 32, Roma
  - orario apertura = lun-sab 9:00-19:



- **2 bibliotecari** (istanze della classe Bibliotecario), uno per il turno del mattino e uno per il turno del pomeriggio, per i quali, per es:

<u>bibliotecario 1:</u>
  - nome = Paolo
  - turno = mattino

<u>bibliotecario 2:</u>
  - nome = Luca
  - turno = pomeriggio

La programmazione orientata agli oggetti

- **un grande numero di libri** (istanze della classe Libro), per i quali, per es:

  libro 1:
  - autore = C.S. Horstmann
  - titolo = Java 2
  - editore = Apogeo
  - collocazione = S21/L303

  libro 2:
  - autore = I. Allende
  - titolo = La casa degli spiriti
  - editore = Feltrinelli
  - collocazione = S13/L44

  ecc...

- **un certo numero di utenti** (istanze della classe Utente), per i quali, per es:

  utente 1:
  - nome = Anna
  - cognome = Goy
  - telefono = 011 1234567

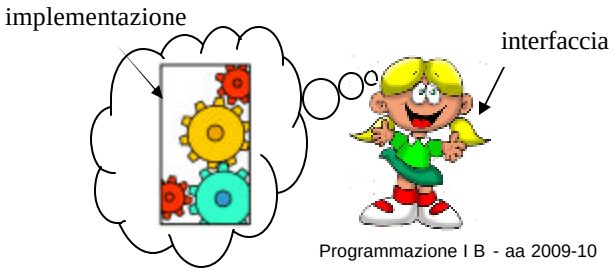
  ecc...

La programmazione orientata agli oggetti

Passo 2: Distinguere interfaccia e implementazione

Quando definisco una **classe** (**tipo**) ne definisco:

- l'**interfaccia** = la "vista esterna" = l'insieme di operazioni che le sue istanze potranno fare
- l'**implementazione** = la "vista interna" = la definizione dei meccanismi che realizzano le operazioni definite nell'interfaccia



La programmazione orientata agli oggetti

Torniamo al nostro **esempio** della biblioteca e consideriamo il Bibliotecario:

- quali **servizi** (**operazioni**) offre al pubblico?

prestito(libro)
restituzione(libro)
prenotazione(libro)
} questi servizi (operazioni) sono **accessibili al pubblico**

= interfaccia

La programmazione orientata agli oggetti

Come li **implementa** (realizza)?

prestito(libro) →

procedura, per trovare il libro, prenderlo, sono "segrete", darlo all'utente, registrare il prestito sulla scheda, **…**
} **non sono visibili al pubblico**

= implementazione

La programmazione orientata agli oggetti

Talvolta si parla di ***design by contract***:

- L'interfaccia può essere vista come un **contratto** tra l'oggetto e il mondo esterno; nel nostro esempio, tra il Bibliotecario e il pubblico (Utente)
- Questo **contratto** definisce **quali servizi** l'oggetto (il nostro Bibliotecario) deve offrire (definisce cioè la sua **interfaccia**), ma **non dice nulla** in merito all'**implementazione** (quali procedure vengono eseguite per realizzare il servizio);
- nel nostro esempio, il Bibliotecario deve garantire i servizi come da contratto, ma può realizzarli come vuole (la realizzazione è una questione privata)

La programmazione orientata agli oggetti

L'**interfaccia** definisce dunque il **comportamento** di un oggetto: nell'esempio, i servizi, cioè le operazioni di *prestito(libro)*, *restituzione(libro)*, *prenotazione(libro)* definiscono il comportamento dei bibliotecari (cioè di tutte le istanze della classe Bibliotecario: Paolo e Luca nell'esempio)

Come avviene l'**interazione** con un oggetto?

- (in linea di massima, vedremo poi le eccezioni) avviene con un'**istanza** (con Paolo o Luca) e non con la classe (non si interagisce con il concetto di Bibliotecario!)
- **inviando un messaggio** all'istanza, con il quale gli si chiede il servizio desiderato; nell'esempio di deve parlare o scrivere a Paolo o Luca per avere un libro in prestito, o per restituirlo, o prenotarlo

La programmazione orientata agli oggetti

ogni biblioteca può avere tanti (*n*) bibliotecari

1 **n**

ogni bibliotecario può appartenere ad una sola biblioteca

1 **m**

ogni biblioteca può avere tanti (*n*) libri

n

ogni libro può appartenere ad una sola biblioteca

ogni biblioteca può avere tanti (*n*) utenti

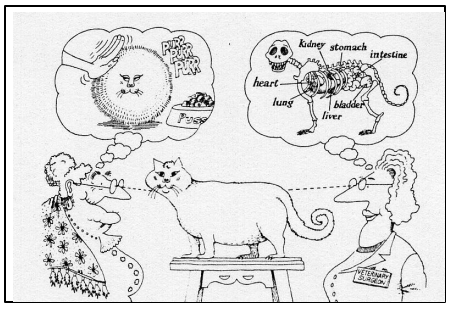
n

ogni utente può essere iscritto a tante (*m*) biblioteche

La programmazione orientata agli oggetti

I principi fondamentali dell'*object-oriented*:

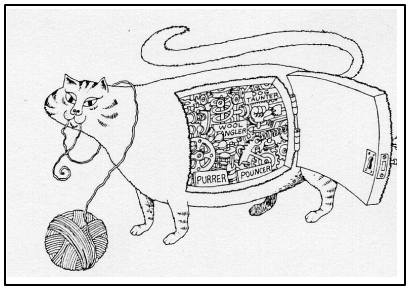
- ***Astrazione***



Un'astrazione rappresenta le caratteristiche essenziali e distintive di un oggetto, dal punto di vista di chi lo guarda
[Grady Booch, *Object Oriented Design*, Benjamin/Cummings, 1991 p. 39]

La programmazione orientata agli oggetti

- ***Incapsulamento*** (*information hiding*)



L'incapsulamento (o *information hiding*) è il principio secondo cui la struttura interna, il funzionamento interno, di un oggetto **non deve essere visibile** dall'esterno

La programmazione orientata agli oggetti

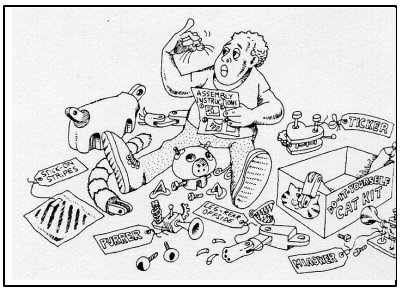
⇒ ogni oggetto è costituito da 2 parti:

- l'*interfaccia* (vista "esterna") → visibile
- l'*implementazione* (vista "interna") → nascosta

L'incapsulamento (o *information hiding*) è il processo che **nasconde** quei dettagli, relativi al funzionamento di un oggetto, che non costituiscono le sue caratteristiche essenziali e distintive [BOOCH, p. 46]

La programmazione orientata agli oggetti

- ***Modularità***



La modularità consiste nella suddivisione di un sistema in una serie di **componenti indipendenti**, che interagiscono tra loro per ottenere il risultato desiderato

scelta dei moduli e delle loro interazioni → definizione dell'architettura del sistema

La programmazione orientata agli oggetti

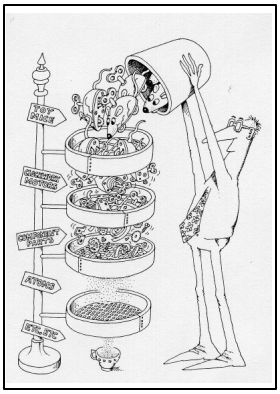
⇒ ogni oggetto (modulo) è **specializzato** in un certo compito: tutti gli altri oggetti (moduli) possono rivolgersi a lui (inviandogli un messaggio) se hanno bisogno dei suoi servizi (in cui lui è specializzato)

Per esempio: pensate a ciò che avviene nella vita reale, in una **società** organizzata come la nostra; all'inizio, per sopravvivere e soddisfare i propri bisogni, ogni uomo cercava di realizzare tutte ciò che gli serviva (costruiva utensili, cacciava, si curava, ecc...).

Con il miglioramento del tenore di vita e la maggiore complessità dei bisogni da soddisfare, come si è organizzata la società? Attraverso la **modularizzazione delle funzioni** e la **collaborazione** di una grande quantità di individui, ognuno specializzato in un compito specifico: se stiamo male ci rivolgiamo ad un medico, per la spesa a verdurieri, macellai, ecc...,

La programmazione orientata agli oggetti

- Struttura gerarchica

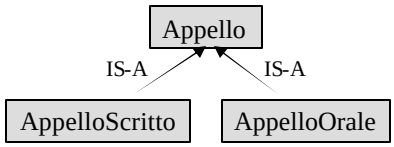


Una gerarchia è un ordinamento di astrazioni [BOOCH, p. 54]

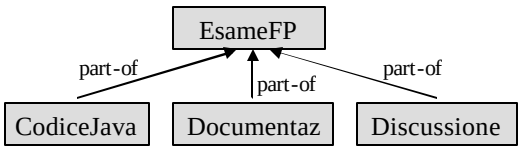
La programmazione orientata agli oggetti

In un sistema complesso, le due principali **gerarchie** sono:

- *kind-of hierarchy* (gerarchia di **classi** e **sotto-classi**)
  Per es.

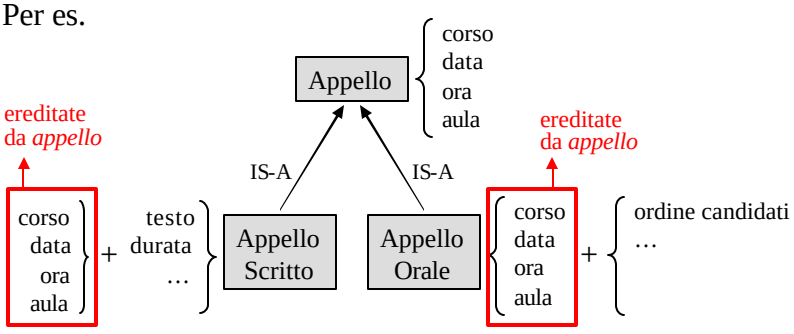


- *part-of hierarchy* (gerarchia di **parti**)
  Per es.



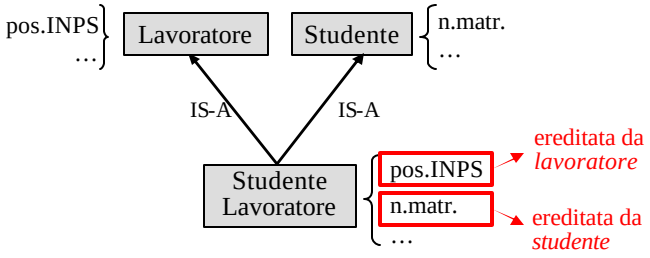
La programmazione orientata agli oggetti

La principale proprietà di una gerarchia *kind-of* è l'**ereditarietà**: se S è una sotto-classe di C, allora S eredita tutte le proprietà di C - ed eventualmente ne definisce altre (*ereditarietà "singola"*)
Per es.



La programmazione orientata agli oggetti

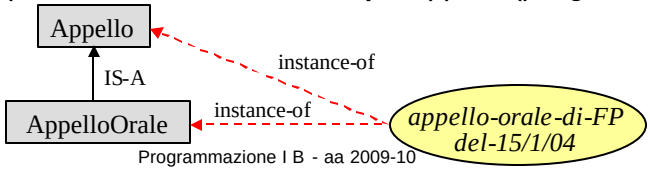
Se una sotto-classe può avere più relazioni *IS-A* (*kind-of*), con classi diverse, allora si ha *ereditarietà "multipla"*: se S è una sotto-classe sia di C1 che di C2, allora S eredita tutte le proprietà di C1 e tutte le proprietà di C2 (ed eventualmente ne definisce altre)
Per es.



La programmazione orientata agli oggetti

In **Java** si ha solo **ereditarietà singola.** In particolare:

- una sotto-classe S eredita dalla sovra-classe C la sua **interfaccia** (cioè l'insieme dei servizi, o operazioni, accessibili dall'esterno)
- un'**istanza** di S sarà automaticamente anche un'**istanza** di C.
  Per es. l'*appello-orale-di-FP-del-15/1/04*, istanza della classe AppelloOrale è anche un'istanza della classe Appello, così come Snoopy, che è un'istanza della classe Cane, è anche un'istanza della classe Mammifero
  - Þ l'istanza *appello-orale-di-FP-del-15/1/04* è di **tipo** AppelloOrale, ma è anche di **tipo** Appello (più generico)

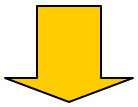


Vantaggi dell' dell'approccio *object-oriented*

- Riuso
- Maggiore leggibilità
- Dimensioni ridotte
- Estensione e modifica più semplici
- Compatibilità
- Portabilità
- Manutenzione del software semplificata
- Migliore gestione del team di lavoro

Riuso

- Approccio procedurale:
  - occorreva conoscere tutto il software
- Approccio ad oggetti:
  - Occorre conoscere l'interfaccia delle classi ma non l'implementazione



  - Molto più conveniente in termini di costi

La programmazione orientata agli oggetti

Alcuni esempi di problemi da risolvere con un approccio object-oriented: analisi e design

1. Gestione dei prestiti in una biblioteca [ok]
2. Gestione delle iscrizioni agli appelli [ok]
3. Gestione dei conti correnti in una banca
4. Gestione dei menu e delle prenotazioni in un ristorante
5. Gestione delle prenotazioni a viaggi organizzati in un'agenzia di viaggi
6. Gestione delle iscrizioni ai corsi in una scuola di lingue

Scegliete uno dei problemi 3-6 e provate a **progettare** un'insieme di classi che lo "risolve" (gestisce)

La programmazione orientata agli oggetti

Esempio: Gestione delle iscrizioni agli appelli

Analisi e Design

1) Quali sono i **tipi di oggetti** (le **classi**) coinvolti nello scenario?

Abbiamo bisogno dei seguenti **concetti** (classi, tipi):

- Segreteria didattica
- Appello
- Studente

Proprietà delle classi I

2) Per ogni **classe** (concetto, tipo) di quali **proprietà** (attributi, caratteristiche) abbiamo bisogno per descriverlo in modo adeguato?

- SegreteriaDidattica
  - corsoDiLaurea
  - listaAppelli [ogni **elemento della lista** sarà un'istanza della classe Appello, cioè sarà **un oggetto di tipo Appello**]
- Appello
  - nomeCorso
  - data
  - aula
  - listaIscritti [ogni **elemento della lista** sarà un'istanza della classe Studente, cioè sarà **un oggetto di tipo Studente**]

Proprietà delle classi II

2) Per ogni **classe** (concetto, tipo) di quali **proprietà** (attributi, caratteristiche) abbiamo bisogno per descriverlo in modo adeguato?

- Studente
  - nome
  - cognome
  - matricola
  - telefono

Operazioni (servizi) offerti I

3) Per ogni **classe** (concetto, tipo) quali sono le **operazioni** pubbliche (i servizi offerti)?

In altre parole, per ogni classe, definisco la sua **interfaccia (operazioni o servizi)**:

- **SegreteriaDidattica**
  - *aggiungi(appello)*
    [*appello* = istanza della classe Appello = oggetto di tipo Appello]
  - *elimina(appello)*
    [*appello* = istanza della classe Appello = oggetto di tipo Appello]
  - *visualizzaAppelli()*

Operazioni (servizi) offerti II

3) Per ogni **classe** (concetto, tipo) quali sono le **operazioni** pubbliche (i servizi offerti)?

In altre parole, per ogni classe, definisco la sua **interfaccia (operazioni o servizi)**:

- **SegreteriaDidattica**
  - *iscrivi(studente, appello)*
    [*studente* = istanza della classe Studente = oggetto di tipo Studente
    appello = istanza della classe Appello = oggetto di tipo Appello]
  - *cancellaIscrizione(studente, appello)*
    [*studente* = istanza della classe Studente = oggetto di tipo Studente
    appello = istanza della classe Appello = oggetto di tipo Appello]
  - *visualizzaIscritti(appello)*
    [*appello* = istanza della classe Appello = oggetto di tipo Appello]

Operazioni (servizi) offerti III

- **Appello**
  - *iscrivi(studente)*
    [*studente* = istanza della classe Studente = oggetto di tipo Studente]
  - *cancellaIscrizione(studente)*
    [*studente* = istanza della classe Studente = oggetto di tipo Studente]
  - *visualizzaIscritti()*
- **Studente**
  - *numeroTelefono()*

Implementazione I

4) Per ogni **classe** (concetto, tipo), per ogni **operazione** definisco la sua **implementazione** (privata), cioè definisco le procedure da eseguire per realizzare le operazioni:

- **SegreteriaDidattica**
  - *aggiungi(appello)*
    ® aggiungi *appello* (che è un'istanza della classe Appello, cioè un oggetto di tipo Appello) a listaAppelli
  - *elimina(appello)*
    ® cancella *appello* (che è un'istanza della classe Appello, cioè un oggetto di tipo Appello) da listaAppelli
  - *visualizzaAppelli()*
    ® stampa a video tutti gli appelli (istanze della classe Appello, cioè oggetti di tipo Appello) contenuti in listaAppelli
  - …

Implementazione II

…

- *iscrivi(studente, appello)*
  ® aggiungi *studente* (che è un'istanza della classe Studente, cioè un oggetto di tipo Studente) a listaIscritti di *appello* (che è un'istanza della classe Appello ed è un elemento di listaAppelli)
  NB Per fare questo invocherà l'operazione *iscrivi(studente)* su *appello*
- *cancellaIscrizione(studente, appello)*
  ® cancella *studente* dalla listaIscritti di *appello* (che è un elemento di listaAppelli)
  NB Per fare questo invocherà l'operazione *cancellaIscrizione(studente)* su *appello*
- *visualizzaIscritti(appello)*
  ® stampa a video tutti gli studenti dalla listaIscritti di *appello* (che è un elemento di listaAppelli)
  NB Per fare questo invocherà l'operazione *visualizzaIscritti()* su *appello*

Implementazione III

...

- **Appello**
  - *iscrivi(studente)*
    ® aggiungi *studente* (che è un'istanza della classe Studente, cioè un oggetto di tipo Studente) a listaIscritti
  - *cancellaIscrizione(studente)*
    ® cancella *studente* (che è un'istanza della classe Studente, cioè un oggetto di tipo Studente) da listaIscritti
  - *visualizzaIscritti()*
    ® stampa a video tutti gli studenti (istanze della classe Studente, cioè oggetti di tipo Studente) contenuti in listaIscritti
- **Studente**
  - *visualizzaTelefono()*
    ® stampa a video tutti il numero di telefono dello studente

La programmazione orientata agli oggetti

Utilizzo del sistema progettato (primo modo)

- ✓ Costruisco un nuova **istanza** (*informatica*) della classe SegreteriaDidattica, fornendo valori specifici per le sue proprietà:
  - corsoDiLaurea = “Informatica"
  - listaAppelli = {}
- ✓ Costruisco un nuova **istanza** (*appFP*) della classe Appello, fornendo valori specifici per le sue proprietà:
  - nomeCorso = "FP"
  - data = "15/1/04"
  - aula = "1.5 PLV"
  - listaIscritti = {}
- ✓ Costruisco un nuova **istanza** (*stud1*) della classe Studente, fornendo valori specifici per le sue proprietà:
  - nome = "Paolo"
  - cognome = "Rossi"
  - matricola = 9875431
  - telefono = 011 1234567

La programmazione orientata agli oggetti

- ✓ **Invoco l'operazione** *iscrivi(stud1)*, definita nell'interfaccia della classe Appello, sull'istanza *appFP*

  NB: L'invocazione dell'operazione *iscrivi(stud1)* corrisponde ad inviare all'istanza *appFP* un **messaggio** in cui si chiede di eseguire l'operazione (iscrivere uno studente)

- ✓ Le **proprietà** (strutture-dati) dell'istanza *appFP* verranno così modificate:
  - o listaIscritti = {*stud1*}

La programmazione orientata agli oggetti

- ✓ **Invoco l'operazione** *aggiungi(appFP)*, definita nell'interfaccia della classe SegreteriaDidattica, sull'istanza *informatica*

  NB: L'invocazione dell'operazione *aggiungi(appFP)* corrisponde ad inviare all'istanza *scidecom* un **messaggio** in cui si chiede di eseguire l'operazione (aggiungere un appello)

- ✓ Le **proprietà** (strutture-dati) dell'istanza *informatica* verranno così modificate:
  - o listaAppelli = {*appFP*}

    NB: *appFP* contiene, in listaIscritti, *stud1*

La programmazione orientata agli oggetti

In alternativa posso (secondo modo)…

- ✓ Costruisco un nuova **istanza** (*informatica*) della classe SegreteriaDidattica, fornendo valori specifici per le sue proprietà:
  - corsoDiLaurea = “Informatica"
  - listaAppelli = {}
- ✓ Costruisco un nuova **istanza** (*appFP*) della classe Appello, fornendo valori specifici per le sue proprietà:
  - nomeCorso = "FP"
  - data = "15/1/04"
  - aula = "1.5 PLV"
  - listaIscritti = {}
- ✓ Costruisco un nuova **istanza** (*stud1*) della classe Studente, fornendo valori specifici per le sue proprietà:
  - nome = "Paolo"
  - cognome = "Rossi"
  - matricola = 9875431
  - telefono = 011 1234567

La programmazione orientata agli oggetti

- ✓ **Invoco l’operazione** *aggiungi(appFP)*, definita nell’interfaccia della classe SegreteriaDidattica, sull’istanza *informatica*

  NB: L’invocazione dell’operazione *aggiungi(appFP)* corrisponde ad inviare all’istanza *informatica* un **messaggio** in cui si chiede di eseguire l'operazione (aggiungere un appello)

- ✓ Le **proprietà** (strutture-dati) dell’istanza *informatica* verranno così modificate:
  - listaAppelli = {*appFP*}

    NB: listaIscritti di *appFP*, per il momento, è vuota!

La programmazione orientata agli oggetti

- ✓ **Invoco l'operazione** *iscrivi(studente, appello)*, definita nell'interfaccia della classe SegreteriaDidattica, sull'istanza *informatica*

  L'implementazione di tale operazione cercherà, allinterno di listaAppelli, l'oggetto corrispondente ad *appello* e su di esso invocherà l'operazione *iscrivi(stud1)*, definita nell'interfaccia della classe Appello

- ✓ Le **proprietà** (strutture-dati) dell'istanza di appello contenuta in listaAppelli verranno così modificate:
  - o listaIscritti = {*stud1*}

NOTA: se immaginiamo uno scenario in cui l'utente (studente) si rivolge alla segreteria didattica per iscriversi agli appelli d'esame, questo secondo modo è più "pulito"

La programmazione orientata agli oggetti

Quando definisco una classe, posso definirla come **sotto classe** di una classe definita in precedenza, per es:

- ✓ Definisco la classe (il tipo) AppelloScritto come sotto classe di Appello

  ⇒ La classe AppelloScritto **eredita** l'interfaccia (e l'implementazione) di Appello

- ✓ Posso **aggiungere** operazioni e proprietà (strutture-dati), per es.
  - **AppelloScritto**
    - o nomeCorso
    - o data
    - o aula
    - o listaIscritti
    - o durata

ereditati da Appello

aggiunto nella sotto-classe

La programmazione orientata agli oggetti

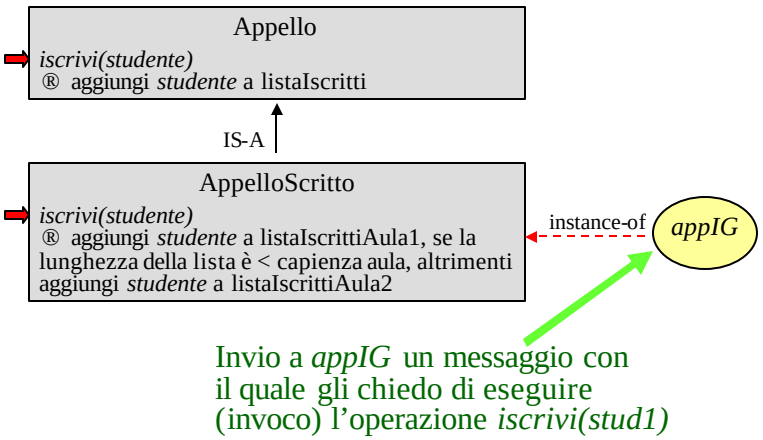
- ✓ Posso **sovrascrivere** (ridefinire) operazioni e proprietà (strutture-dati), per es:
  - Al posto di un'unica listaIscritti definisco 2 liste:
    - AppelloScritto
      - nomeCorso
      - ...
      - listaIscrittiAula1
      - listaIscrittiAula2
  - **Ridefinisco** le operazioni *iscrivi(studente)*, *cancellaIscrizione(studente)* e *visualizzaIscritti()*:
- AppelloScritto
  - *iscrivi(studente)*
    ® aggiungi *studente* a listaIscrittiAula1, se la lunghezza della lista è < capienza aula, altrimenti aggiungi *studente* a listaIscrittiAula2
  - *cancellaIscrizione(studente)*
    ® cancella *studente* da listaIscrittiAula1 o listaIscrittiAula2
  - *visualizzaIscritti()*
    ® stampa a video tutti gli studenti contenuti in listaIscrittiAula1 + listaIscrittiAula2

La programmazione orientata agli oggetti

Utilizzo del sistema progettato

- ✓ Costruisco un nuova **istanza** (*appIG*) della classe AppelloScritto, fornendo valori specifici per le sue proprietà:
  - nomeCorso = "IG"
  - data = "15/1/04"
  - aula = "1.3 PLV"
  - listaIscrittiAula1 = {}
  - listaIscrittiAula2 = {}
  - durata = "1 ora"
- ✓ **Invoco l'operazione** *iscrivi(stud1)* sull'istanza *appIG*
- Quale operazione viene eseguita?
- Quella definita nella classe AppelloScritto (che aggiungerebbe *stud1* a listaIscrittiAula1), oppure quella definita nella classe Appello (che aggiungerebbe *stud1* a listaIscritti)?

La programmazione orientata agli oggetti



Invio a *appIG* un messaggio con il quale gli chiedo di eseguire (invoco) l'operazione *iscrivi(stud1)*

Quale operazione viene eseguita, quella definita nella classe AppelloScritto (che aggiungerebbe *stud1* a listaIscrittiAula1), oppure quella definita nella classe Appello (che aggiungerebbe *stud1* a listaIscritti)?

La programmazione orientata agli oggetti

In questo caso, viene eseguita l'operazione definita nella sotto-classe AppelloScritto

- le **proprietà** (strutture-dati) dell'istanza *appIG* verranno così modificate:
  - listaIscrittiAula1 = {*stud1*}

NB: se l'operazione non fosse stata ridefinita, l'invocazione di *iscrivi(stud1)* su *appIG* utilizzerebbe la definizione **ereditata** da Appello (la quale modifica listaIscritti...)!