Programmazione I

A.A. 2002-03

Programmazione Orientata agli Oggetti:

Classi Astratte e Interfacce (Lezione XXIX)

Prof. Giovanni Gallo
Dr. Gianluca Cincotti
Dipartimento di Matematica e Informatica
Università di Catania
e-mail : {gallo, cincotti}@dmi.unict.it

A volte i nostri oggetti sono vaghi…

Esempio:

Definiamo la classe dei *"poligoni"*.

Un metodo desiderabile per tale classe è quello che ne calcola e fornisce il *perimetro*.

Non potendo però essere più precisi sulla natura dei poligoni stessi (quanti lati ha un poligono?) come implementare un tale metodo?

Un dilemma!

- siamo ad un dilemma:
  - vogliamo sottolineare la necessità di un metodo "perimetro" per ogni oggetto della classe dei "poligoni“ e delle sue sottoclassi;
  - non possiamo implementare alcun metodo "perimetro" senza prima specializzare in qualche modo la classe dei poligoni.

La soluzione di JAVA

JAVA consente al programmatore mentre crea la classe dei “**`poligoni`**” di “fare promessa" di implementare in ogni sottoclasse dei “**`poligoni`**” un opportuno metodo “**`perimetro()`**”.

La classe “**`poligoni`**” deve però essere dichiarata con il qualificatore “**`abstract`**”. La classe creata sarà dunque una **classe astratta.**

JAVA pretende e vigila che la promessa sia mantenuta! Il programmatore “deve mantenere la promessa" quando implementa le sottoclassi di una classe astratta.

Classi "astratte“, in pratica (1)

Precisiamo meglio:

una classe qualificata come "abstract" nella sua dichiarazione può avere metodi di due tipi:

- metodi “standard” perfettamente definiti e implementati;
- metodi "abstract" di cui si fornisce solo la intestazione (nome, tipo restituito, parametri) ma non si fornisce alcuna implementazione o body.

esempio1.java

Classi Astratte, in pratica (2)

La responsabilità di fornire un "body" adeguato per i metodi abstract ricade quindi su chi implementa le classi derivate.

Esse DEBBONO provvedere tale implementazione, pena un errore di compilazione.

(JAVA si ricorda delle promesse e le vuol mantenute!)

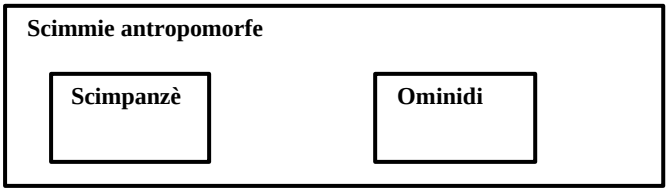
ATTENZIONE: non è possibile costruire (istanziare) usando il comando new un oggetto di una classe astratta.
Infatti quale codice dovrebbe essere legato al metodo “abstract” di tale oggetto?
E’ possibile istanziare solo le classi “concrete” derivate da una abstract.

Eredità singola

Tutti gli esempi visto ad adesso sono esempi di **eredità singola**:

Ogni sotto-classe estende (o eredita) una sola super-classe.

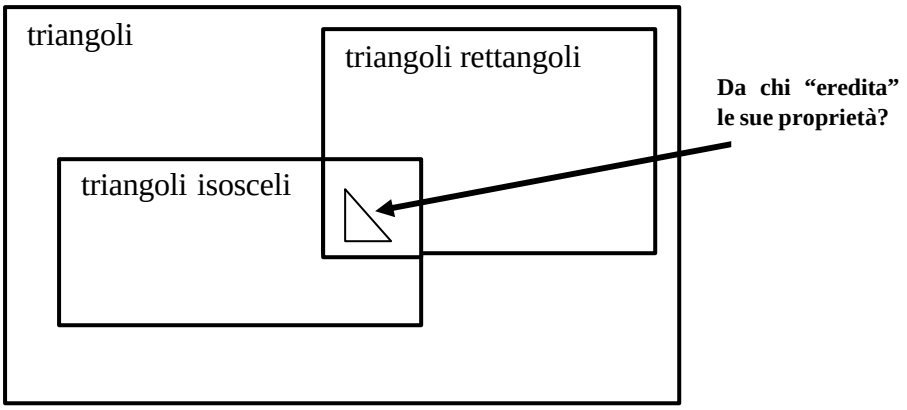
Esempio:



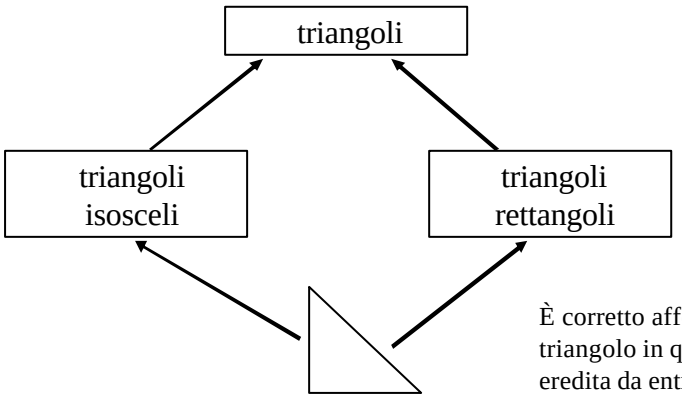
Sebbene potente questo meccanismo non è sufficiente in talune situazioni:

Eredità Multipla 1

Esempio:



Eredità multipla 2



È corretto affermare che il triangolo in questione eredita da entrambi.

Una situazione comune

Abbiamo costruito una classe "studente" che gestisce bene gli studenti (per esempio materie, frequenze eccetera)

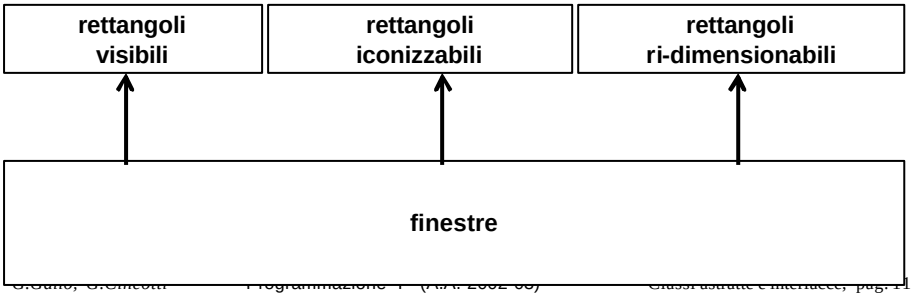
Abbiamo costruito una classe "atleta" che gestisce bene gli atleti di una squadra (partite, allenamenti, ruoli, eccetera)

Ci serve scrivere un programma per una **squadra di atleti universitari**… ereditarla da "studente" o da "atleta"?

Altro esempio di eredità multipla

Una finestra, elemento grafico del sistema operativo, è (contemporaneamente):

- Un rettangolo visibile,
- Un rettangolo "iconizzabile",
- Un rettangolo ri-dimensionabile.



JAVA ammette solo eredità SINGOLA

Questo è un limite che i creatori di JAVA si sono imposti per semplificare il lavoro di javac.

Il C++ ammette eredità multiple.

Come ovviare al problema?

JAVA ha una tecnica per aggirare l'ostacolo che fa uso delle "interfaces".

*Oltre alle classi: le **interfacce***

La soluzione di JAVA al problema è usare le interfacce.

- Esse non sono tecnicamente classi, ma oggetti pubblici.
- Una interfaccia non ha variabili di istanza.
- Una interfaccia può contenere la dichiarazione di costanti.
- Una interfaccia ha solo dichiarazioni di metodi pubblici senza body (cioè non implementate).

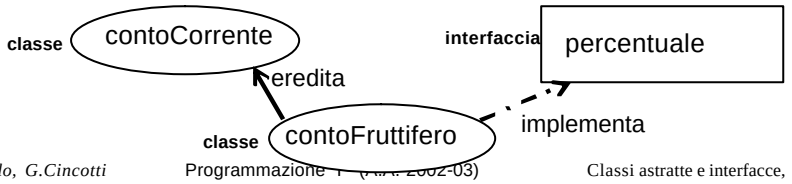
In pratica una interfaccia contiene le informazioni su cosa è necessario (in termini di metodi) per garantire certe funzionalità.

Un esempio (giocattolo): contoCorrente

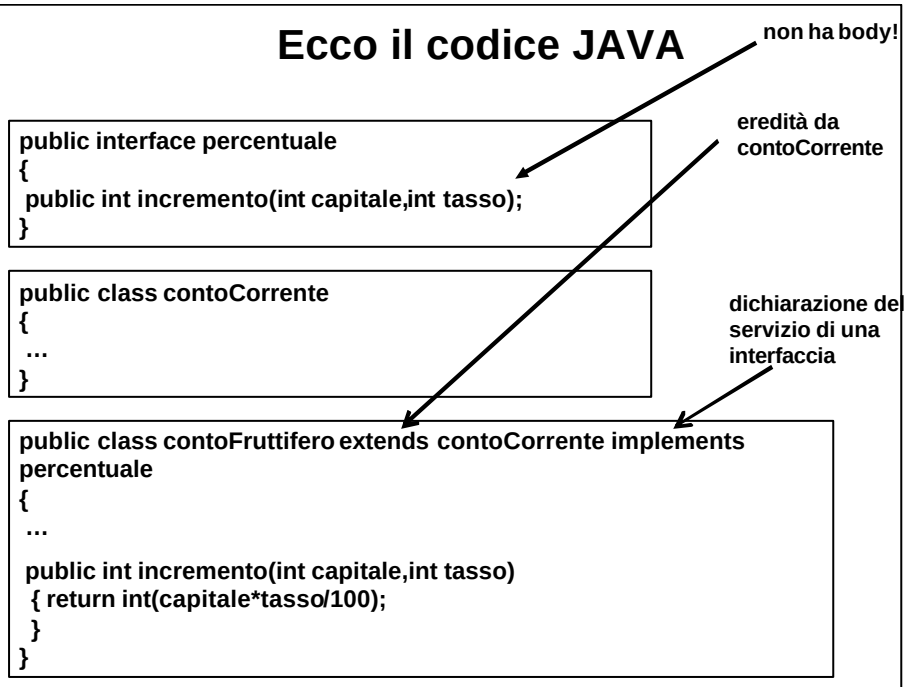
Supponiamo di definire una classe contoCorrente.
Essa avrà variabile di istanza: int saldo;
Essa ha metodi per dire se il conto è in rosso, o in positivo, e metodi per incrementare o decrementare il conto stesso.

Supponiamo di derivare da essa una classe contoFruttifero.
Essa ha gli stessi metodi e campi di contoCorrente ma ha anche un metodo per incrementare il saldo di un valore percentuale.
Le occorre dunque implementare una funzionalità "*calcolo percentuale*".

Tale funzionalità è in realtà utile a molte classi… essa può essere descritta in una interfaccia.



Ecco il codice JAVA



Proprietà delle interfacce

- Le interfacce sono **sempre pubbliche** e tutti i loro metodi sono pubblici (usare il qualificatore "public" è quindi sovrabbondante ma suggerito per chiarezza).
- Le interfacce **non possono essere istanziate** in oggetti!!
- Si possono implementare in una classe **più di una** interfaccia.
- Le **classi astratte somigliano alle interfacce** ma non consentono l'eredità multipla.
- Si possono creare **gerarchie ereditarie di interfacce.**

Un esempio di uso facile (e comodo) di interfaccia

Supponiamo si debba implementare tutta una serie di classi che debbono usare tutte le medesime costanti.

Si possono raccogliere le costanti tutte in una interfaccia SENZA metodi.

Basterà dire che ciascuna classe "implements" la interfaccia per avere a disposizione di ciascuna classe tali costanti senza doverle dichiarare più volte.

Per esempio per una serie di classi per una contabilità:

```
public interface costanti_bilancio
{ public static final int TASSO_DEBITORI=10;
  public static final int TASSO_CREDITORI=11;
}
```

Ogni classe che dichiarerà di implementare costanti_bilancio avrà automaticamente a disposizione le costanti appena definite.

Vedere gli esempi:

- esempio2.java
- esempio3.java

Fine