UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Informatica
@UnivAQ

Laboratorio di Algoritmi e Strutture Dati
a.a. 2022/2023

INTERFACCE (CONTINUA)

Giovanna Melideo
Università degli Studi dell'Aquila
DISIM

Ricapitoliamo

- Abbiamo già visto come usare una **classe astratta** (classe **`VettoreOrdinabile`**) per scrivere un algoritmo utilizzabile per ordinare oggetti di una qualsiasi classe
- Con questo approccio è necessario dichiarare una classe specializzata per ogni classe che si intende ordinare, anche se tale classe contiene poco codice
  - es. classe `VettoreIntero`
  - es. classe `VettorePunto`
  - es. classe `VettorePersona (homework!)`

Verso le interfacce

- Cosa hanno in comune la classe **`Punto`** e la classe **`Integer`** ?
  - Gli oggetti di entrambe possono essere ordinati secondo un criterio univoco
- In effetti, si può desiderare di ordinare oggetti di un gran numero di classi, secondo criteri diversi
- Sarebbe comodo dire che una qualsiasi classe è ordinabile se ha un metodo **`confronta`** che consenta di confrontare due istanze e di stabilire quale delle due deve precedere l'altra.

L'interfaccia `Ordinabile` (1 di 4)

- In questo modo delineiamo una specie di **classe trasversale** che accomuna classi diverse la cui unica caratteristica comune è quella di avere un metodo con la stessa firma.
- Potremmo poi avere una classe che ordina questa classe trasversale, senza la necessità di avere classi specializzate.

L'interfaccia `Ordinabile` (2 di 4)

Per esempio:

```
interface Ordinabile {
    public int confronta (Ordinabile obj);
}
```

- **Ricorda**: non si può creare un'istanza di un'interfaccia con l'istruzione new, visto che sarebbe vuota, per cui quando un oggetto è di un tipo corrispondente a un'interfaccia, significa che appartiene a una classe che implementa quell'interfaccia.

L'interfaccia `Ordinabile` (3 di 4)

- Una classe può implementare una o più interfacce dichiarandole esplicitamente e implementando i metodi dichiarati nelle interfacce stesse.
- In tal caso, gli oggetti di questa classe saranno riconosciuti anche come oggetti che implementano l'interfaccia.

L'interfaccia `Ordinabile` (4 di 4)

- Modifichiamo la classe **`VettoreOrdinabile`** vista precedentemente in modo che possa funzionare con le interfacce.
- rif. **`Ordinabile`**

Tipi di dato astratto - cenni (ADT)

Un tipo di dato astratto (ADT) definisce una **categoria concettuale** con le sue proprietà:

- una definizione di tipo
  - implica un dominio, D
- un insieme di operazioni ammissibili su oggetti di quel tipo
  - funzioni: calcolano valori sul dominio D
  - predicati: calcolano proprietà vere o false su D

Tipi di dato astratto e interfacce

- Nei linguaggi O.O. come Java, i tipi di dato astratti corrispondono alle interfacce
- un'interfaccia **descrive un comportamento** che sarà assunto da una classe che realizza l'interfaccia, nel senso che per ogni classe che implementa un'interfaccia l'utilizzatore può:
  - creare un oggetto della classe ("oggetto" corrisponde ad un "valore" del dominio D)
  - invocare i metodi pubblici della classe ("metodo pubblico" corrisponde a "operazione ammissibile")

Tipi di dato astratto e strutture dati

- Una struttura dati è la **realizzazione concreta** (o implementazione) di un ADT
  - un ADT definisce **cosa si può fare** con una struttura dati che realizza l'interfaccia (categoria concettuale)
  - la classe che rappresenta concretamente la struttura dati definisce invece **come vengono eseguite** le operazioni
- In altre parole valgono le associazioni:

ADT – Interfaccia

Struttura dati – Classe

Ereditarietà multipla (1 di 3)

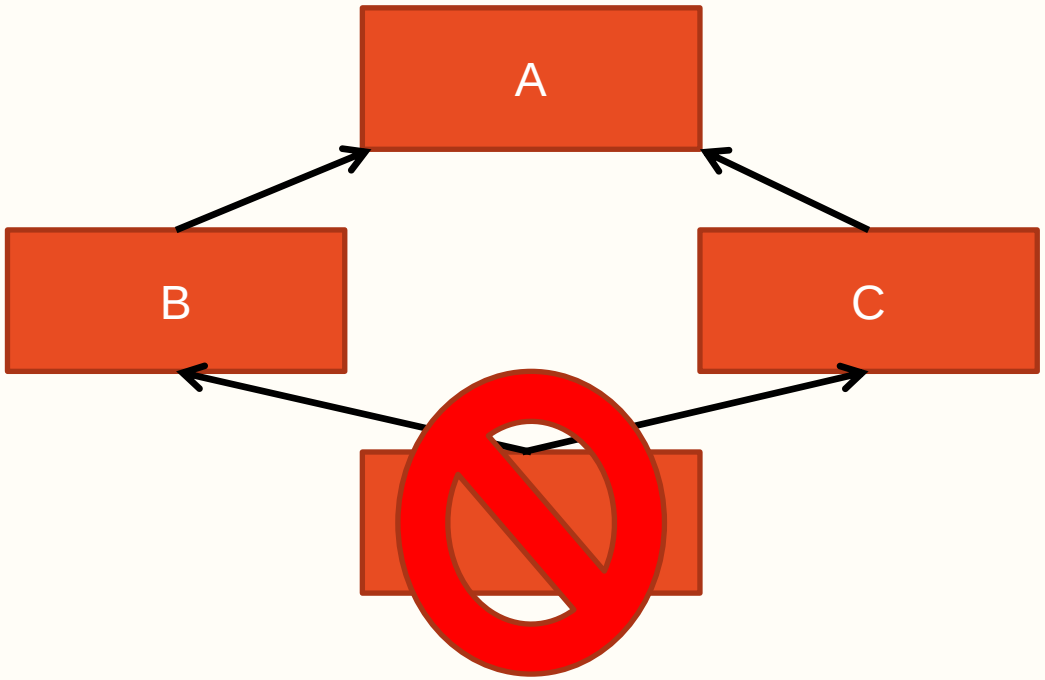
- In Java non esiste la cosiddetta “ereditarietà multipla” (come in C++)
- In pratica non è possibile scrivere:

```
public class Idrovolante extends Nave, Aereo {
        . . .
}
```

- Questa permette ad una classe di estendere più classi contemporaneamente

Ereditarietà multipla (2 di 3)

- Problema del diamante



Ereditarietà multipla (3 di 3)

- L'ereditarietà multipla può essere causa di ambiguità:
  - due classi B e C ereditano dalla classe A
  - la classe D eredita sia da B che da C
  - se un metodo in D chiama un metodo definito in A, da quale classe viene ereditato?
  - in particolare, cosa succede se B e C presentano due differenti implementazioni di uno stesso metodo?

Ereditarietà multipla ed interfacce

- Tale ambiguità prende il nome di **problema del diamante**, proprio a causa della forma del diagramma di ereditarietà delle classi, simile ad un diamante.
- In Java per risolvere questo inconveniente si è adottato questo compromesso:
  - una classe può estendere una sola classe alla volta, cioè ereditare i dati ed i metodi effettivi da una sola classe base
  - può invece implementare infinite interfacce, simulando di fatto l'ereditarietà multipla, ma senza i suoi effetti collaterali negativi.

Classi astratte vs Interfacce (1 di 4)

- Il vantaggio che offrono sia le classi astratte che le interfacce, risiede nel fatto che esse possono “obbligare” le sottoclassi ad implementare dei **comportamenti**
- Una classe che eredita un metodo astratto infatti, deve fare override del metodo ereditato oppure essere dichiarata astratta.
- Dal punto di vista della progettazione quindi, questi strumenti supportano l’astrazione dei dati.

Classi astratte vs Interfacce (2 di 4)

- Un'evidente differenza pratica è che possiamo simulare l'ereditarietà multipla solo con l'utilizzo di interfacce.
- Tecnicamente la differenza più evidente è che un'interfaccia non può dichiarare né variabili ne metodi concreti, ma solo costanti statiche e pubbliche e metodi astratti.
- È invece possibile dichiarare in maniera concreta un'intera classe astratta (senza metodi astratti). In quel caso il dichiararla astratta implica comunque che non possa essere istanziata.

Classi astratte vs Interfacce (3 di 4)

- Quindi una classe astratta solitamente non è altro che un'**astrazione troppo generica per essere istanziata** nel contesto in cui si dichiara.
- Un'interfaccia invece, solitamente non è una vera astrazione troppo generica per il contesto, ma semmai una "**astrazione comportamentale**", che non ha senso istanziare in un certo contesto.

Classi astratte vs Interfacce (4 di 4)

- Le classi astratte pure definiscono un legame più forte con la classe derivata poiché ne rappresentano il **tipo base** definendone il comportamento comune
- Le interfacce possono invece essere usate per definire un **modello generico**, che implementa un comportamento comune a classi di vario genere e natura

UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Informatica
@UnivAQ

Domande?

Giovanna Melideo
Università degli Studi dell'Aquila
DISIM