UNIVERSITÀ
DI PARMA

oop: ereditarietà

Alberto Ferrari

ereditarietà

- una nuova classe (**classe derivata**) viene creata a partire da una classe esistente (**classe base**)
- la classe derivata **eredita** le variabili membro e le funzioni membro della classe base
- la classe derivata può **aggiungere** variabili membro e funzioni membro

ridefinizione delle funzioni membro ereditate

- una classe derivata può ***cambiare la definizione*** di una funzione membro ereditata
- in questo caso la definizione della classe derivata deve contenere la dichiarazione della funzione membro ereditata
- possiamo avere ereditarietà per
  - ***estensione***
    (***aggiunta*** di nuove variabili e/o funzioni)
  - ***ridefinizione***
    (***overriding*** di funzioni)

ereditarietà e riuso

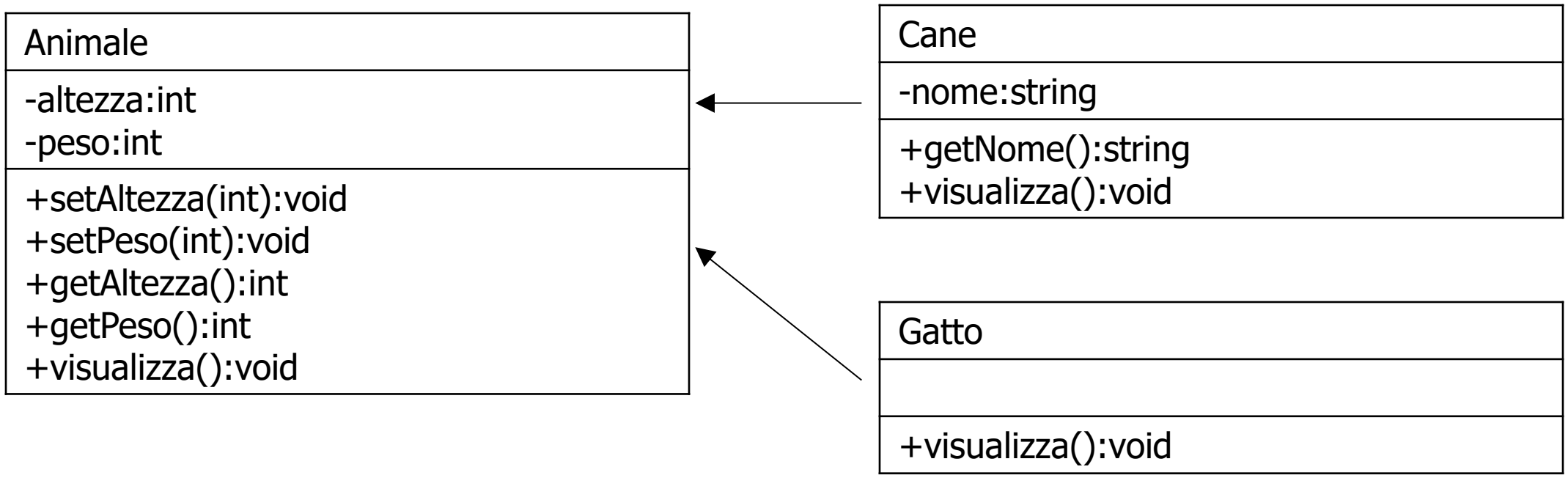
- la classe base contiene il ***codice comune*** alle classi derivate
- l'ereditarietà consente di ***riusare*** il codice della classe base

Un esempio senza ereditarietà

Animale
-altezza:int -peso:int
+setAltezza(int):void +setPeso(int):void +getAltezza():int +getPeso():int +visualizza():void

Cane
-altezza:int -peso:int -nome:string
+setAltezza(int):void +setPeso(int):void +getAltezza():int +getPeso():int +getNome():string +visualizza():void

Un esempio con ereditarietà



Le classi della gerarchia in C++
Animale


```
class Animale
{
public:  Animale( const int = 0, const int = 0 );
    void setAltezza( int );
    int getAltezza();
    void setPeso( int );
    int getPeso();
    void visualizza();
private:
    int altezza;
    int peso;
};
```

Le classi della gerarchia in C++
Cane


```
class Cane: public Animale
{
public:
     Cane( const int = 0 , const int = 0, string = "Bill");
     void visualizza();
     void setNome( string );
private:
     string nome;
};
-------------------------------------------------------------
Cane::Cane( const int a, const int p, string n ): Animale( a, p)
{
    setNome( n );
} …
void Cane::visualizza()
{  cout << "Sono un cane di nome: " << nome << endl;  Animale::visualizza();
}
```

Le classi della gerarchia in C++
Gatto


```
class Gatto: public Animale
{
public:
    Gatto ( const int = 0, const int = 0);
    void visualizza();
};

Gatto::Gatto( const int a, const int p)   : Animale(a, p)
{}
void Gatto::visualizza()
{
   cout << "Sono un gatto";
   Animale::visualizza();
}
```

costruttori nelle classi derivate

- un costruttore della classe base ***non viene ereditato***
- può essere ***invocato*** nella definizione del costruttore della classe derivata per inizializzare le variabili ereditate
- se non è invocato, il costruttore di ***default*** della classe base viene invocato ***automaticamente***

ordine di chiamata dei costruttori

- la chiamata del costruttore della classe base è la prima azione del costruttore della classe derivata
- se A→B→C quando viene creato un oggetto di classe C prima viene chiamato un costruttore della classe A, poi un costruttore della classe B, poi vengono intraprese le rimanenti azioni del costruttore di classe C

uso dei membri privati della classe base

- i ***membri privati*** della classe base ***non sono referenziabili*** nelle definizioni delle funzioni membro della classe derivata
  - verrebbe violato il principio di ***incapsulamento***
- le funzioni membro della classe derivata possono accedere alle variabili membro private della classe base tramite le funzioni ***accessor*** e ***mutator*** *(se presenti)*
- le ***funzioni membro private*** della classe base ***non*** sono ***accessibili*** *(di fatto non sono ereditate)*

Il qualificatore protected

- una variabile o funzione membro qualificata come ***protected*** può essere referenziata nelle funzioni membro di una classe derivata
- le variabili membro ***protected*** agiscono come se fossero ***protected*** in ogni classe derivata
- molti ritengono che l'uso di variabili membro *protected* ***comprometta*** l'***incapsulamento***
- è buona norma utilizzare protected ***solo*** quando assolutamente ***necessario***

ridefinizione (overriding) e sovraccarico (overloading)

- una funzione ***ridefinita*** in una classe derivata ha lo ***stesso numero e tipo di parametri*** della funzione della classe base (***overriding***)
- una funzione ***sovraccaricata*** in una classe derivata ha un ***diverso numero e/o tipo di parametri*** rispetto alla funzione della classe base e la classe derivata ha entrambe le funzioni (***overloading***)

accesso a una funzione della classe base ridefinita

- una classe derivata può ridefinire una funzione della classe base
- è possibile invocare su un oggetto della classe derivata la ***versione*** della funzione data nella ***classe base***
- si utilizza l'operatore ::, che in questo caso è ***obbligatorio***, altrimenti la funzione chiamante continuerebbe in realtà a chiamare se stessa generando un loop

relazione “is a”

- un ***oggetto*** di una classe ***derivata*** può essere usato ***ovunque*** può essere usato un ***oggetto*** della classe ***base***
- un oggetto di una classe derivata ha ***più di un tipo***
- ***Cane is a Animale***

funzioni che non vengono ereditate

- oltre alle funzioni membro private ***non vengono ereditati***
  - ***costruttori***
  - ***distruttori***
  - ***costruttori di copia***
  - ***operatori di assegnamento***
- se non vengono definiti vengono creati quelli di ***default***

distruttori in classi derivate

- quando il ***distruttore*** di una ***classe derivata*** è invocato, viene invocato ***automaticamente*** il distruttore della ***classe base***
- se A→B→C, quando termina lo scope di un oggetto di classe C viene chiamato prima il distruttore della classe C, poi quello della classe B, infine quello della classe A
- i distruttori sono chiamati in ordine inverso rispetto ai costruttori

relazioni tra oggetti

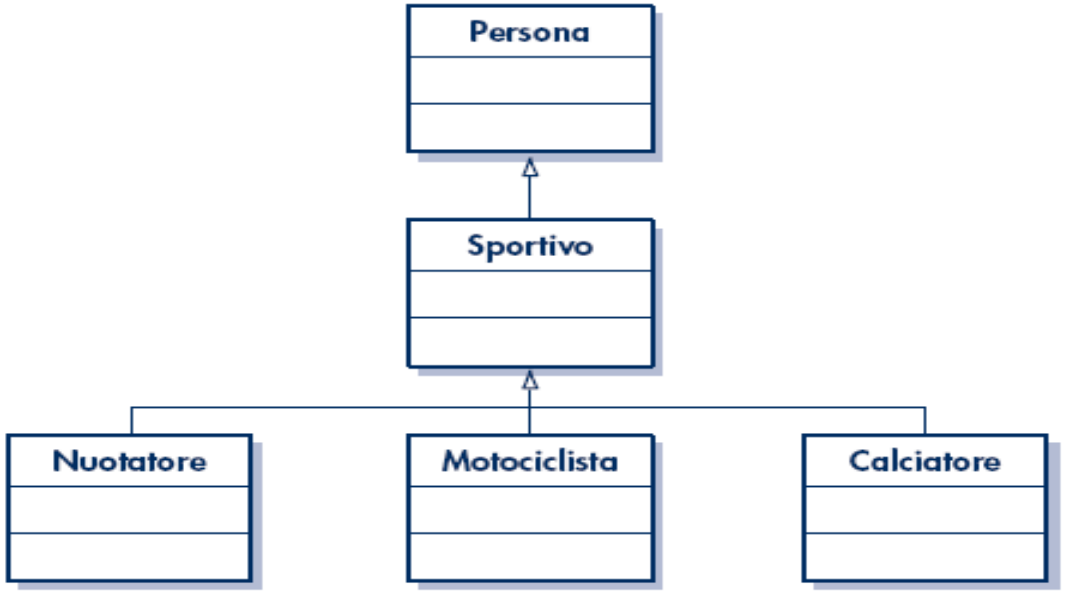
- relazione "***is a***"
  - esempio: un Gatto ***is a*** Animale
- relazione "***has a***"
  - esempio: an Computer ***has a*** Processore

ereditarietà protetta e privata

- ***ereditarietà protetta***: i membri pubblici della classe base sono protetti nella classe derivata quando sono ereditati
- ereditarietà privata: nessun membro della classe base può essere referenziato nella classe derivata
  - la relazione “is a” non è valida
  - sono raramente usate

gerarchia di classi

- l'ereditarietà può estendersi a più livelli generando quindi una ***gerarchia di classi***
- una classe derivata può, a sua volta, essere base di nuove sottoclassi
- ***Sportivo*** è *sottoclasse* di ***Persona*** ed è *superclasse* di ***Nuotatore***, ***Motociclista*** e ***Calciatore***
- nella parte alta della gerarchia troviamo le ***classi generiche***, scendendo aumenta il ***livello di specializzazione***



ereditarietà multipla

- una classe derivata può avere ***più di una classe base***
- possono esserci situazioni ***ambigue***
- richiede una ***conoscenza approfondita*** del linguaggio
- in ***alcuni linguaggi*** (es Java) ***non è ammessa*** l'ereditarietà multipla