Laboratorio di programmazione

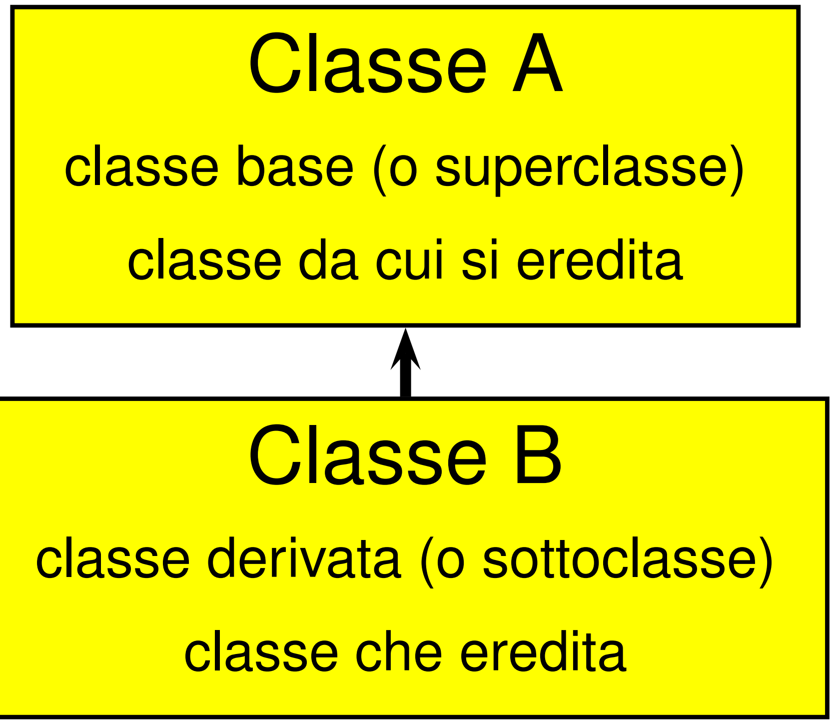
Lezione XII

Tatiana Zolo

zolo@cs.unipr.it

EREDITARIETÀ

Proprietà fondamentale dei linguaggi object-oriented.



EREDITARIETÀ

La classe derivata programma solo quegli aspetti che differiscono da o estendono il comportamento della classe base. Può:

- ➜ aggiungere nuovi campi e nuovi metodi a quelli ereditati;
- ➜ ridefinire alcuni dei metodi ereditati;
- ➜ cambiare la visibilità di ciò che ha ereditato.

Ereditarietà

La classe derivata programma solo quegli aspetti che differiscono da o estendono il comportamento della classe base. Può:

- ➜ aggiungere nuovi campi e nuovi metodi a quelli ereditati;
- ➜ ridefinire alcuni dei metodi ereditati;
- ➜ cambiare la visibilità di ciò che ha ereditato.

Visibilità: anche la classe derivata non può vedere la parte privata della classe base, vede soltanto la parte pubblica $\Longrightarrow$ nuovo livello di accesso: protected.

I tre livelli di accesso (che una classe permette a chi sta fuori) sono:

- ➜ private: visibile *solo* ai metodi della classe;
- ➜ protected: visibile ai metodi della classe e delle classi derivate (violazione all'incapsulamento, usare solo dove realmente necessario);
- ➜ public: visibile a tutti.

Ereditarietà

Attenzione: la qualifica `private/protected/public` (usata nel corpo della classe) non dice nulla su cosa la classe derivata decida di rendere a sua volta visibile di ciò che ha ereditato. La definizione della classe derivata deve specificare quale visibilità applicare agli attributi ereditati. Sintassi:

```
class B : [private | protected | public] A {
  // corpo della classe B.
}
```

- ➜ `private/protected/public` dentro il corpo della classe stabiliscono *cosa sia visibile a chi* fuori dalla classe.
- ➜ `private/protected/public` nella dichiarazione di una classe derivata stabiliscono *cosa questa classe rende visibile agli altri* di ciò che ha ereditato.

Ereditarietà

DERIVAZIONE	EFFETTO
public	la visibilità resta come è public → public, protected → protected, private → private
protected	la visibilità è ristretta a protected *public* → *protected*, protected → protected, private → private
private	la visibilità è ristretta a private *public* → *private*, *protected* → *private*, private → private

EREDITARIETÀ: ESEMPIO

Il metodo `area()` deve essere ridefinito in ogni sottoclasse.

```
class Poligono {
  public:
    void area() { cout << ''area poligono'' << endl }
}
class Rettangolo : public Poligono {
  public:
    void area() { cout << ''base * altezza'' << endl }
}
class Triangolo: public Poligono {
  public:
    void area() { cout << ''base * altezza / 2'' << endl }
}
```

EREDITARIETÀ

- Cosa si eredita?
  - tutti i campi dati (anche quelli private a cui la classe derivata non potrà accedere direttamente);
  - tutti i metodi (anche quelli private a cui la classe derivata non potrà accedere direttamente).
- Cosa NON si eredita?
  - costruttori;
  - distruttore;
  - operatore di assegnamento.

EREDITARIETÀ

- Costruttore: al momento dell'invocazione del costruttore della classe derivata si ha una chiamata automatica al costruttore di default della classe base (invocato sempre *prima* del costruttore della classe derivata).

```
Base::Base(int x) {
  a = x;
}
Derivata::Derivata(int x, int y, int z)
  : A(x) {
  b = y;
  c = z;
}
```

- Distruttore: come i costruttori, ma l'ordine di invocazione è invertito. Il distruttore della classe base viene invocato *dopo* l'esecuzione del distruttore della classe derivata (se quest'ultimo è stato definito).

ESERCIZI

1. Riprendere la classe Tempo della lezione VIII tramite la quale si esprime il concetto di *orario*: aggiungervi la ridefinizione dell'operatore '==' e dell'operatore '<'. Modellare ora il concetto di *orario con data* con la nuova classe Data_Tempo derivata dalla classe Tempo.

```
class Data_Tempo : public Tempo {
public:
  Data_Tempo();
  Data_Tempo(int anno, int mese, int giorno,
      int ora, int min, int sec);
  int giorno() const;
  int mese() const;
  int anno() const;
  Data_Tempo operator+(const Tempo& T) const;
  Data_Tempo operator<(const Data_Tempo& T) const;
private: int gg; int mm; int aa; };
```

ESERCIZI

- ➜ `operator+`: somma un oggetto di tipo Tempo ed un oggetto di tipo Data_Tempo. Se la somma porta ad un valore maggiore di 24 ore allora deve essere aggiornata anche la data.