



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Ereditarietà

Stefano Ghidoni



Agenda

- Ereditarietà / gerarchie di classi
- Un esempio
- Classi astratte (1)
- Slicing



Descrivere entità geometriche

- Come descrivere entità geometriche?
- Vogliamo creare una libreria per descrivere entità geometriche
 - Punti
 - Forme
 - ...
- Che classi creare? Come rappresentare nel codice le relazioni che esistono tra le varie forme?



Entità da rappresentare

- Come descrivere entità geometriche?
- Una forma possiede:
 - Alcune caratteristiche comuni ad altre forme
 - Descritta da un insieme di punti
 - Funzionalità disegno
 - Alcune caratteristiche che dipendono dalla specifica forma
 - Non è possibile specificare finché parliamo di forma in senso generale



Criteri di progettazione

- È possibile raggruppare le caratteristiche comuni
 - Classe base
- Le classi più specifiche *ereditano* tali caratteristiche comuni, e le estendono
- Una scelta progettuale usata spesso è produrre:
 - Molte classi
 - Con poche operazioni



Point

- L'entità base è il punto
 - Definito come struct: ogni valore è ammissibile
 - Raggruppa le due coordinate

```
struct Point {  
    int x, y;  
};
```



Shape

- Definiamo ora una classe Shape che rappresenta una forma *generica*
 - Costruita usando Point
- Molte operazioni sono definibili per una forma generica
- Shape implementa **un'interfaccia** comune a tutte le forme
 - L'effettiva implementazione dipende dalla specifica forma
 - **Definisco già tutte le operazioni che voglio permettere/fare per le classi derivate**



Classe Shape

```
class Shape {  
public:  
    void draw() const;  
    virtual void move(int dx, int dy);  
  
    Point point(int i) const;           // accesso ai punti  
    int number_of_points() const;  
  
    Shape(const Shape&) = delete;  
    Shape& operator=(const Shape&) = delete;  
  
    virtual ~Shape() { }  
  
    // ...
```

L'utente può disegnare, spostare, accedere a un punto in lettura, sapere quanti punti ci sono



Classe Shape

```
protected: Vuol dire che gli elementi sono accessibili dalle
Shape() { } classi che ereditano, ma non dalle altre
Shape(initializer_list<Point> lst);

virtual void draw_lines() const;           // disegno
void add(Point p);                      // aggiunge un punto
void set_point(int i, Point p); // points[i] = p;

private:
vector<Point> points;      // non usato da tutte le Shape
Color lcolor;                // dalla libreria grafica
Line_style ls;               // dalla libreria grafica
Color fcolor;                // dalla libreria grafica
};
```

- Nota: line color vs fill color

L'utente non può creare oggetti di classe Shape, non può disegnare(_linee), aggiungere un punto o inizializzare punti



Classi astratte

- Shape è una classe astratta
 - Costruttore di default e con inizializzazione sono **protected** (in questo caso, come se fossero **private**)

protected:

```
Shape() { }  
Shape(initializer_list<Point> lst);
```

Quindi non si possono creare oggetti di questa classe.

La classe base può essere un elemento della gerarchia, che può avere senso a sé stante.

Altrimenti può essere un'interfaccia. Shape di per sé non è una forma di qualche tipo, ma un concetto astratto di forma. Qualunque forma effettivamente disegnabile è sia una forma, sia qualcos'altro.

Shape è una forma non disegnabile, ma comunque comune a tutte le forme.



Classi astratte

- Shape è una classe astratta
 - Costruttore di default e con inizializzazione sono protected (in questo caso, come se fossero private)
- Esiste un altro modo (più frequentemente usato) per definire le classi astratte
 - Funzioni virtuali pure (vedi oltre)
- Classe astratta: è lo strumento che implementa un'interfaccia



Ereditarietà

- È possibile definire classi che ereditano dalla classe shape

```
class Circle : public Shape {  
    // ...  
};
```

Il simbolo che permette di ereditare
Definisce il tipo di ereditarietà

- La classe Circle è derivata di Shape
 - Cosa eredita?



Ereditarietà

- Ereditarietà public
 - La classe derivata può accedere ai **membri public e protected** della classe base
- Altri tipi di ereditarietà:
 - Private e protected (raramente usati)

→ Ad esempio, Circle ha `lcolor`, ma *non* vi può accedere



"Is a"

- Qual è il rapporto tra una classe base e la sua derivata?
- Relazione "Is a"
- Un oggetto di classe derivata è *un (is a)* oggetto di classe base
 - Es: un cerchio è *una* forma geometrica
- Questa relazione non è biunivoca (una forma non è un cerchio)



"Has a"

- Un'altra possibile relazione tra classi è "has a"
- Qualche esempio?
- *Has a* definisce la composizione tra classi (un oggetto di una classe contenuto in un'altra classe)
 - Es: un'automobile *ha un* volante



Slicing

- La relazione *is a* rende possibili alcune operazioni problematiche
- Es: potrei inserire un oggetto circle in un vector di Shape
 - Quali potenziali problemi?

```
void my_fct(const Circle& c)
{
    vector<Shape> v;
    v.push_back(c);
}
```

Si può aggiungere un Circle in un vector di Shape, visto che Circle *is a* Shape2



```
void my_fct(const Circle& c)
{
    vector<Shape> v;
    v.push_back(c);
}
```

- Se Circle è derivata di Shape, probabilmente ha dati membro aggiuntivi
 - Per esempio: Circle ha un raggio
- Utilizzare il costruttore di copia (tramite il `push_back`) causerebbe uno **slicing**



Slicing

- Lo slicing si verifica quando tentiamo di inserire una classe derivata (es., circle) in uno slot che riesce a contenere solo i membri della classe base
- Il compilatore risolve *affettando* l'oggetto, lasciando solo i membri che trovano posto



- Costruttore di copia e operator= sono disabilitati (`delete`)
 - Evita la creazione di costruttore di copia e operator= di default
 - Altrimenti, problemi di **slicing**

```
// Reminder
class Shape {
public:
// ...
    Shape(const Shape&) = delete;
    Shape& operator=(const Shape&) = delete;
// ...
```



- Costruttore di copia e operator= sono disabilitati (delete)
 - Evita la creazione di costruttore di copia e operator= di default
 - Ora la seconda riga causa errore di compilazione

```
void my_fct(const Circle& c)
{
    vector<Shape> v;
    v.push_back(c); // errore: costruttore di copia
                    // disabilitato
}
```



Disabilitare la copia

- "Basically, class hierarchies plus pass-by-reference and default copying do not mix" (BS)
- Classi create per funzionare come classi base di una gerarchia richiedono di **disabilitare**:
 - Costruttore di copia (copy constructor)
 - Assegnamento di copia (copy assignment)
 - Non è necessario disabilitare move perché se non viene previsto, il compilatore non lo creerà



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Ereditarietà

Stefano Ghidoni



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE