32. I metodi (alcune note)

Corso di Informatica

Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
21/12/2020

Outline

- Funzioni **`inline`**
- Funzioni **`const`**
- Funzioni **`static`**
- Parola chiave **`friend`**

Funzioni `inline` (1)

- Una funzione dichiarata con il qualificatore **`inline`** offre una forma di ottimizzazione
- Nella pratica, ogni chiamata ad una funzione **`inline`** viene sostituita in fase di compilazione con il corpo della funzione stessa

```
int main()
{
    Vettore v1 = new Vettore(10.0, 45.0);
    cout << v1.getModulo() << endl;
    return 0;
}
```



```
int main()
{
    Vettore v1 = new Vettore(10.0, 45.0);
    cout <<
        v1.inline double Vettore::getModulo() {
            return this->modulo;
        }
        << endl;
    return 0;
}
```

Funzioni `inline` (2)

- Per definire una funzione **`inline`**, possiamo definirla solo una volta nell'header, oppure in ogni file sorgente nella quale viene invocata
  - *Questo provoca riscrittura di porzioni anche abbondanti di codice, oppure un'aperta violazione dei principi dell'incapsulamento, per cui le funzioni* **`inline`** *vanno utilizzate solo nel caso in cui si abbia evidenza empirica di un effettivo aumento delle prestazioni*

```
// Definizione nell'header
// geometria.h
class Vettore {
    // ...
    public:
        inline double getModulo() {
            return this->modulo;
        }
}
```

```
// Definizione nei sorgenti
// geometria.h
class Vettore {
    // ...
    public:
        double getModulo();
}

// geometria.cpp
// Da ridefinire in ogni sorgente!
inline double Vettore::getModulo() {
    return this->modulo;
}
```

Funzioni `const`

- Una funzione dichiarata con il qualificatore **`const`** non può modificare gli attributi dell'istanza della classe
  - *Può comunque accedervi in lettura!*
- Per questo motivo, i getter sono di solito decorati con questo qualificatore

```
// geometria.h
class Geometria::Vettore
{
    //...
    private:
        double modulo;
        double angolo;
    public:
        double getModulo() const;
        double getAngolo() const;
        // ...
}
```

```
// geometria.cpp
double Vettore::getModulo() const {
    return this->modulo;
}

double Vettore::getAngolo() const {
    return this->angolo;
}
```

Funzioni `static` (1)

- Abbiamo visto come il qualificatore **`static`** si riferisca a variabili il cui tempo di visibilità coincide con l'intera esecuzione del programma
- Nella OOP, ciò si ripercuote sul concetto di classe ed istanza: funzioni (e membri) **`static`** vivono quindi 'indipendentemente' dalle istanze di classe
- Si dice infatti che **funzioni e membri statici si riferiscono alla classe**, mentre **funzioni e membri non statici si riferiscono all'istanza**
- Va da sé che è possibile modificare un membro dichiarato come **`static`** esclusivamente usando una funzione **`static`**

Funzioni `static` (2)

- Supponiamo ad esempio di voler inserire un riferimento all'origine dello spazio cartesiano, comune per tutti i nostri vettori

```
// geometria.h
class Punto {
    // ...
}

class Vettore {
    private:
        static Punto origine;
    public:
        static Punto getOrigine();
}
```

```
// geometria.cpp
Punto Vettore::getOrigine()
{
    return origine;
}
```

La parola chiave `friend` (1)

- Il modificatore **`friend`** può essere applicato a funzioni o classi che, pur non facendo parte dell'ambito della classe cui si riferiscono, possono accedere ad attributi protetti e privati
- Aimmaginiamo che una classe **`SpazioVettoriale`** voglia accedere ai dettagli interni dei vettori componenti la sua base. Potremo usare il modificatore **`friend`**:

```
// geometria.h
class Vettore {
    private:
        // ...
        friend class SpazioVettoriale;
}
```

Domande?

42