29. Le Classi

Corso di Informatica

Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
14/11/2020

Outline

- Il concetto di classe
- Definizione ed implementazione
- Utilizzo
- Costruttori
- Distruttori

Il concetto di classe

- Permette di creare nuovi tipi
- Ogni nuovo tipo ha **attributi** e **metodi**, ognuno dei quali **accessibile secondo un opportuno modificatore**
- Sfrutta i meccanismi di accesso per definire una **interfaccia** verso il mondo esterno (**information hiding**)

Definizione ed implementazione (1)

- Definiamo una classe partendo dall'header
- Iniziamo con la classe **`PersonaBase`** (il perché di questo nome sarà chiaro in seguito)
- L'header **`persona.h`** (continua…):

Include guards →

```
#ifndef PERSONA_H
#define PERSONA_H
#include <string>

using namespace std;
```

Namespaces →

```
namespace Persona
{
    class PersonaBase;
}
```

Definizione ed implementazione (2)

- L'header **`persona.h`** (continua…):

Attributi →

Metodi →

```
class Persona::PersonaBase
{
private:
    string nome;
    string cognome;
    int eta;
public:
    string getNome();
    string getCognome();
    int getEta();
    void setNome(string nuovoNome);
    void setCognome(string nuovoCognome);
    void setEta(int nuovaEta);
};

#endif // !PERSONA_H
```

Definizione ed implementazione (3)

- Di norma, gli attributi sono dichiarati **`private`**, mentre i metodi **`public`**
- In questo modo, vi è un solo modo di accedere o modificare gli attributi, ovvero mediante i metodi
- La convenzione (non stringente!) dice che i metodi che iniziano con il prefisso **`get`** sono i metodi **accessori** (**getter**), mentre quelli che iniziano con il prefisso **`set`** sono i metodi **modificatori** (**setter**)
- Le implicazioni sono importanti: **abbiamo un'unica fonte di verità, definita all'interno del metodo!**
- **Per cambiare il comportamento dobbiamo modificare solo i metodi**

Definizione ed implementazione (4)

- L'implementazione è contenuta in **`persona.cpp`** (continua):

Include guards ⟶

```
#include "persona.h"
#include <iostream>
```

Namespaces ⟶

```
using namespace std;
using namespace Persona;
```

Definizione ed implementazione (5)

- L'implementazione è contenuta in **`persona.cpp`** (continua):

Getter

```
string PersonaBase::getNome() {
    return nome;
}

string PersonaBase::getCognome() {
    return cognome;
}

int PersonaBase::getEta() {
    return eta;
}
```

Definizione ed implementazione (6)

- L'implementazione è contenuta in **`persona.cpp`**:

Setter 

```
void PersonaBase::setNome(string nuovoNome) {
    if (nuovoNome.length() > 2) {
        nome = nuovoNome;
    }
}

void PersonaBase::setCognome(string nuovoCognome) {
    if (nuovoCognome.length() > 2) {
        cognome = nuovoCognome;
    }
}

void PersonaBase::setEta(int nuovaEta) {
    if (nuovaEta >= 0) {
        eta = nuovaEta;
    }
}
```

Definizione ed implementazione (7)

- L'implementazione dei setter ci mostra la potenza dell'information hiding
- Ad esempio, nel metodo **`setEta`**, controlliamo che il nuovo valore dell'età sia maggiore di zero
- Se accedessimo direttamente all'attributo, **dovremmo implementare questi controlli ad ogni accesso!**
  - *È evidente come la situazione possa diventare rapidamente ingestibile all'aumentare delle dimensioni della codebase*

```
void PersonaBase::setEta(int nuovaEta) {
    if (nuovaEta >= 0) {
        eta = nuovaEta;
    }
}
```

Utilizzo

- Le classi si usano allo stesso modo delle normali variabili. Ad esempio:

```
int main() {
    PersonaBase truce;
    truce.setNome("Truce");
    truce.setCognome("Baldazzi");
    truce.setEta(18);

    cout << "Nome: " << truce.getNome() << "\tCognome: «
         << truce.getCognome() << "\tEta': »
         << truce.getEta() << endl;

    return 0;
}
```

Costruttori (1)

- Sono delle particolari funzioni che servono a creare una **istanza** di una classe
- Hanno delle caratteristiche ben definite, ovvero:
  - *hanno lo stesso nome della classe;*
  - *non hanno alcun tipo o valore di ritorno*
- Ne esiste almeno uno per classe (**costruttore di default**)
  - *Il compilatore lo specifica per noi nel caso lo si ometta*
  - *È però consigliabile definirlo manualmente, a meno che non si tratti di classi molto semplici*
- Di solito si usano costruttori parametrizzati in overloading
- Il consiglio è di usare i setter definiti in precedenza per rispettare l'applicazione delle tecniche di information hiding

Costruttori (2)

```
// persona.h
public:
    PersonaBase();
    PersonaBase(string nuovoNome, string nuovoCognome, int nuovaEta);

// persona.cpp
PersonaBase::PersonaBase() {
    setNome("Non definito");
    setCognome("Non definito");
    setEta(0);
}

PersonaBase::PersonaBase(string nuovoNome, string nuovoCognome, int nuovaEta) {
    setNome(nuovoNome);
    setCognome(nuovoCognome);
    setEta(nuovaEta);
}
```

Distruttori (1)

- Servono a liberare le risorse allocate da un oggetto quando questo esce dal suo ambito di visibilità
- Come i costruttori, hanno delle caratteristiche ben definite, ovvero:
  - *hanno lo stesso nome della classe, preceduto dalla tilde (~);*
  - *non hanno alcun tipo o valore di ritorno*
- Ne esiste **solo** uno per classe
- Nel caso di classi molto semplici, è possibile ometterlo
- Va però **sempre** specificato nel caso di uso di reference (puntatori) o accesso a risorse (come stream) all'interno della classe
  - *Si può usare l'operatore* `delete` *nel primo caso, od opportuni metodi (ad esempio, l'equivalente* `iostream` *di* `fclose()`*) nel secondo*

Distruttori (2)

```
// persona.h
// ...
class Persona::PersonaBase
{
    private:
        // ...
        int* puntatoreEta;
        // ..
    public:
        // ...
        ~PersonaBase();
        // ...
}
```

```
// persona.cpp
PersonaBase::PersonaBase()
{
    // ...
    puntatoreEta = &eta;
    // ...
}
PersonaBase::~PersonaBase()
{
    delete puntatoreEta;
}
```

Domande?

42