Appunti di Programmazione a Oggetti

4 ottobre 2022

Rosso Carlo

Contents

1 Introduzione

Un linguaggio di programmazione si dice ad oggetti se gode di tre proprietà:

- incapsulamento;
- ereditarietà;
- polimorfismo.

L'incapsulamento è l'inaccessibilità dell'implementazione di una classe o di una funzione. Questa proprità permette di rendere il codice più leggibile e più mantenibile. Le classi per esempio sono implementazioni del tipo di dato astratto. Con l'incapsulamento la definizione di classe viene nascosta. In questo modo si opera con un sistema a scatola nera: non sai come i dati sono rappresentati, pensi solo a come utilizzare la tecnologia, senza curarti del perchè funzioni in questo modo. NB Nel corso di programmazione ad oggetti impariamo come nascondere l'implementazione, in modo che altri utilizzino l'implementazione in modo chiaro senza conoscerla.

Il c++ offre una programmazione generica: proprio come in rust, ci sono classi e funzioni generiche; inoltre offre la gestione degli errori (non abbiamo ancora chiarito bene).

1.1 Operatori

Il c++ fornisce molti operatori, circa 50. Un operatore è una fuzione che viene chiamata attraverso dei simboli o attraverso la ripetizione di simboli.

Per esempio, l'operatore "¡¡", oppure l'operatore "::", scoping.

1.2 namespace

Una condizione frequente nella stesura di programmi lunghi riguarda l'inquinamento dello spazio del nome. Il nome delle variabili deve essere significativo e preferibilmente in inglese.

Il namespace è l'equivalente del modulo su rust. Permette di raggruppare (recintare) le keyword in modo che siano accessibili solo attraverso l'operatore di scoping, oppure attraverso le keyword "using namespace NomeSpazio".

```
namespace SpazioUno {
    namespace f {
        g()
    }
    ...
}
```

Il namespace viene dichiarato come nell'esempio.
Nel momento in cui vari namespace sono contenuti uno dentro l'altro e noi desideriamo accedere solo ad un namespace particolare è utile ricorrere alla tecnica dell'alias:

```
namespace Uno = SpazioUno;
namespace ...
```

Questa tecnica si utilizza nel file in cui utilizziamo le classi o le funzioni contenute in un determinato namespace.
Se intendiamo utilizzare tutte le funzioni in un namespace allora possiamo evitare di indicare ogni volta il namespace a cui desideriamo accedere:

```
// using namespace SpazioUno; // deprecato
// using namespace SpazioUno::f; // per includere l'intero namespace f
using namespace SpazioUno::f::g; // forma preferibile
```

Tra le due righe di codice riportate, la prima è deprecata, perchè si rischia di inquinare lo spazio dei nomi attuale. Inoltre, se si utilizzano solo alcune funzioni in un namespace è preferibile incorporare solo quelle funzioni, per leggibilità e per evitare di inquinare lo spazio dei nomi.

1.3 ADT

L'abstract data type è alla base dell'incapsulamento: viene nascosta l'implementazione ed è visibile solo l'interfaccia, ovvero i metodi pubblici e le operazioni proprie.

1.4 Classe

Il c++ fornisce l'astrazione classe. L'astrazione classe è l'astrazione di un tipo, si tratta dell'implementazione dell'ADT che viene fornita in c++. Prendiamo per esempio una classe che rappresenta il tempo, se sceglessimo di divere l'ora in ore, minuti, secondi, ci ritroveremmo ad utilizzare 12 byte: un byte per ciascuno di questi campi. Un sistema più conveniente consiste nel contare i secondi passati dalla mezzanotte, in questo modo manteniamo tutte le informazioni necessarie per conoscere l'ora, ma occupiamo solo un byte.
Implementiamo la classe orario:

```
\label{classe_orario}
classe orario {
    private:
        int sec; // il campo che indica i secondi dalla mezzanotte

    public:
        orario();
        orario(int, int);
        orario(int, int, int);
        int Ore();
        int Minuti();
        int Secondi();
};
```

Una classe definisce un proprio namespace!
Il this è l'equivalente di self in rust. Il this è un handel all'oggetto che viene modificato in un metodo. Poniamo il caso in cui un metodo ritorni un'oggetto della classe, una soluzione è la seguente:

```
classe A {
    public:
        A f() {
            return this;
        }
};
```

Il this serve per applicare una funzione all'oggetto di invocazione.

In line

Le definizioni delle funzioni possono essere fatte inline:

```
classe orario {
    ...
    Ore() {
        return sec / (60 * 60);
    }
    ...
};
```

Se la definizione è fatta inline allora quando è chiamato il metodo (una funzione relativa ad una classe), il corpo della funzione è inserito al posto della funzione. Come lato negativo questo comporta un codice molto più pesante e lungo (da evitare).

Costruttore

Un costruttore è una funzione che per nome ha il nome della classe di cui è il costruttore. Il costruttore senza parametri è chiamato default contructor; se non si specifica il costruttore di default allora viene generato un costruttore standard che per generare un nuovo oggetto chiama i costruttori di default di ciascun campo della classe. Come vediamo nell'esempio **??**, facciamo un overloading del costruttore in modo da avere costruttori diversi in base ai dati che abbiamo disponibili, senza dover inquinare lo spazio dei nomi.

1.5 information hiding

Information hiding consiste nell'evitare dipendente del codice per qualcosa che potrebbe cambiare. In questo modo l'implementazione diventa indipendente dall'utilizzo di una classe, per cui è più semplice cambiare una classe senza dover riscrivere il codice che utilizza quella classe.
L'information hiding è permesso dai specificatori di accesso: public e private. Tutti i metodi e i campi presenti in private sono accedibili solo da metodi della classe. Mentre qualunque campo o metodo in public è accedibile.
In genere la documentazione descrive il funzionamento dell'interfaccia pubblica, mentre l'interfaccia privata viene tralasciata per evitare dipendenze dal codice che potrebbero essere modificato.

1.6 The rule of three

Se definiamo uno dei seguenti, allora dovremmo definirli tutti e tre:

- distruttore;
- costruttore di copia;
- operatore di assegnazione di copia;

Questa regola è anche chiamata la regola del pollice (rule of thumb).

1.7 Array

Gli array hanno dimensione statica: il valore deve essere conosciuto a compilazione.

Nella costruzione di un array i valori sono assegnati mediante costruttori di copia:

```
class C {
    public:
        int i;
        C(int x=3): i(x) {}
        ~C() {std::cout << i << "~C ";}
};

int main() {
    C a[4] = {C(1), C(), C(8)};
    // distruzione oggetti temporaneri, stampa: 8~C 3~C 1~C
    std::cout << a[0].i << a[1].i << a[2].i << a[3].i << std::endl;
    //stampa: 1383
}
//a all'uscita stampa 3~C 8~C 3~C 1~C
```

Con il file "*.h" l'informazione da nascondere (come sono implementate le classi) viene mostrato: è un rischio perchè ti espone ad attacchi esterni. Soluzione:

```
// file "C_handle.h"
class C_handle {
    public:
        // parte pubblica
    private:
        class C_privata;    // dichiarazione incompleta
        C_privata* punt;    // solo punt. e ref. per dich. incompleta
};
```

```
// file "C_handle.cpp"
class C_handle::C_privata {
    ...
};
```

1.8 Friend function

Una funzione amica (si utilizza la keyword "friend") può eccedere ai campi privati e protetti di una classe diversa. Rende qualche funzione dipendente dalla classe (rischia di diventare un problema di dipendenza).