



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Progettare un'interfaccia: funzioni const e helper function

Stefano Ghidoni



Agenda

- Riprendiamo l'esempio Date
- Funzioni const
- Funzioni membro vs helper function



Richiamo – Date

- Date definita in 05_3:

```
class Date {  
    int y, m, d;  
  
public:  
    Date (int y, int m, int d);  
    void add_day(int n);  
    int month() { return m; }  
    int day() { return d; }  
    int year() { return y; }  
};
```



Funzioni membro const

- Richiamo: argomenti passati per reference e per const reference

```
void some_function(Date& d, const Date& start_of_term)
{
    int a = d.day(); // ok
    int b = start_of_term.day(); // dovrebbe essere ok
    d.add_day(3); // ok
    start_of_term.add_day(3); // errore
}
```

È un problema? Perché?



Funzioni membro const

- La chiamata a Month::day() applicata a un const Date& non garantisce che l'oggetto non sarà modificato
 - day() in effetti non modifica l'oggetto, ma ciò non è esplicito
- Soluzione: dichiarare day() come **funzione membro const**

Una funzione costante è una funzione che non applicherà modifiche che lo dichiara



Funzioni membro const

```
class Date {  
public:  
    // ...  
    int day();  
    Month month();  
    int year();  
  
    void add_day(int n);  
    void add_month(int n);  
    void add_year(int n);  
    // ...  
private:  
    int y;  
    Month m;  
    int d;  
};
```



Funzioni membro const

```
class Date {  
public:  
    // ...  
    int day() const;  
    Month month() const;  
    int year() const;  
  
    void add_day(int n);  
    void add_month(int n);  
    void add_year(int n);  
    // ...  
private:  
    int y;  
    Month m;  
    int d;  
};
```

Si può anche fare l'overloading di una funzione con versione const e non const



Funzioni membro const

- Modificare un membro in una funzione const causa un errore di compilazione

```
int Date::day() const
{
    ++d;          // errore! Modifica in una funzione const
    return d;
}
```

Funzioni membro
vs
helper function



Funzioni membro vs helper function

- Abbiamo visto che per alcune funzionalità possiamo scegliere
 - Helper function
 - Member function
- Come effettuare la scelta?

Una helper function a cui si vuole dare il permesso di accedere ai dati membri si dichiara "friend".



Funzioni membro vs helper function

- "A function that can be simply, elegantly, and efficiently implemented as a freestanding function (that is, a non-member function) should be implemented outside the class. That way, a bug in that function cannot directly corrupt the data in a class object." (BS)
- In caso di dati membro corrotti, solo le funzioni membro devono essere controllate



Funzioni membro vs helper function

- Quante funzioni sono ragionevoli per la classe Date?
- "A few years ago I surveyed a number of commercially used Date libraries and found them full of functions like `next_sunday()`, `next_workday()`, etc. Fifty is not an unreasonable number for a class designed for the convenience of the users rather than for ease of comprehension, implementation, and maintenance." (BS)

50 funzioni membro rendono il debugging complesso, infatti il codice può essere esteso, quindi anche il problema può essere esteso.



Helper function

- Abbiamo già visto il passaggio da helper function a funzioni membro
 - Abilita la protezione dei dati
- Ora riscopriamo il ruolo delle helper function
 - La protezione dei dati membro si ottiene con le funzioni membro
 - Dati meglio protetti se le funzioni membro **sono poche**



Helper function

- Le helper function utilizzano spesso argomenti della classe di cui sono helper
 - Non sempre, non una regola!
- operator== e operator != sono tipiche helper function
- Spesso le helper function sono inserite in un namespace assieme alla classe

```
namespace Chrono
{
enum class Month { /* ... */ }
class Date { /* ... */ }
Date next_sunday(const Date& d) { /* ... */ }
Date next_workday(const Date& d) { /* ... */ }
// ...
}
```



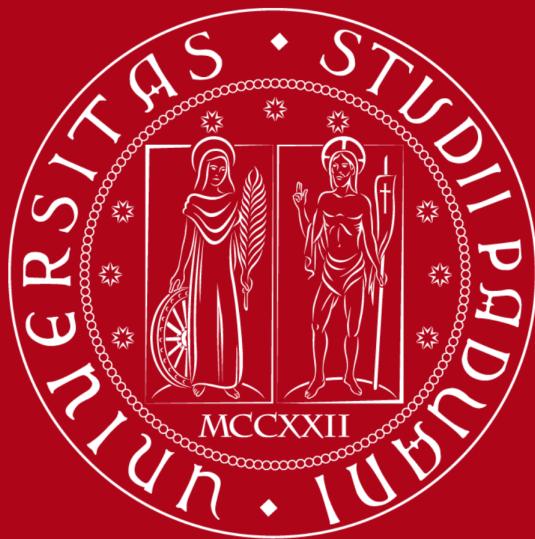
Helper function

```
Date next_sunday(const Date& d)
{
    // accedere utilizzando d.day(), d.month(), d.year()
    // creare una nuova Date da ritornare
}

Date next_weekday(const Date& d) { /* ... */ }
bool leapyear(int y) { /* ... */ }

bool operator==(const Date& a, const Date& b)
{
    return a.year() == b.year()
        && a.month() == b.month()
        && a.day() == b.day();
}

bool operator!=(const Date& a, const Date& b)
{ return !(a==b); }
```



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Progettare un'interfaccia: funzioni const e helper function

Stefano Ghidoni