

# Classi, oggetti e template

Anna Corazza

aa 2023/24

# Dove studiare

- ▶ Str'13, parte III

**Str'13** Bjarne Stroustrup, The C++ Programming Language (4th edition), 2013

<https://www.stroustrup.com/4th.html>

# Classi

- ▶ Si accede ai membri di una classe mediante il punto(.) per gli oggetti e la freccia per i puntatori (->).
- ▶ In una classe si possono definire degli operatori.
- ▶ Una classe è un namespace contenente i suoi membri.
- ▶ Tipi e alias di tipi possono essere membri di una classe (si pensi ad un `enum`, per esempio).
- ▶ I membri possono venir dichiarati:
  - `public` : interfaccia;
  - `private` : implementazione.
- ▶ Una `struct` può venir vista come una `class` in cui tutti i membri sono `public`.

# Semplice esempio

---

```
class Eempio{
private:
    int m;
public:
    //Costruttore: inizializza m
    Eempio(int i=0):m{i} {}
    int metodo(int i){
        int old=m;
        m=i;
        return old;
    }
};

Eempio var {7};

int user(Eempio var, Eempio* ptr){
    int x=var.mf(7);
    int y=ptr->mf(9);
    int z=var.m; // errore!
```

---

# Copia di oggetti

- Per default gli oggetti possono venir copiati: quindi un'istanza di una classe può venir inizializzata con una copia di un'altra istanza:

---

```
class Date{
    int d,m,y;
public:
    void init(int dd, int mm, int yy);

    void add_year(int n);
    void add_month(int n);
    void add_day(int n);
}
```

```
Date my_birthday;
my_birthday.init(14,1,1988);
Date d1 = my_birthday;
```

---

# Copia di oggetti

## Element-wise

- ▶ Per default, la copia di un oggetto corrisponde alla copia di ogni singolo membro.
- ▶ Se si vuole che il comportamento sia diverso da questo, è necessario definire opportunamente la classe.
- ▶ Questo naturalmente vale anche per le istanze di una classe: l'assegnamento significa la copia dell'istanza.
- ▶ Anche qui, copia significa copia di ogni singolo membro, ma se si vuole un comportamento diverso basta ridefinire l'operatore di assegnamento

# Controllo dell'accesso

- ▶ In C++, ci sono diversi livelli di protezione. Per il momento:  
    `private` default se sono all'inizio; ma possono essere  
        anche alla fine, e allora la parola chiave  
        `private` è necessaria  
    `public`

# Costruttori

- ▶ Quando la classe ha un costruttore, esso viene chiamato ogni volta che viene creato un nuovo oggetto.
- ▶ Un costruttore può avere parametri di ingresso.

---

```
class Date{
    int d,m,y;
public:
    Date(int dd, int mm, int yy);
    // .
};
Date today=Date(23,6,1983);
Date today(23,6,1983); // lo stesso, ma abbreviato
Date my_birthday; // errore: mancano i parametri di
                   ingresso
Date release1_0(10,12) // errore, manca un parametro
```

---



# Costruttore

- Visto che il costruttore inizializza gli oggetti della classe, possiamo usare la notazione propria delle inizializzazioni -:

---

```
Date today=Date {23,6,1983};  
Date christmas {25,12,1993};
```

---

- Di solito abbiamo più costruttori, con signature diverse.
- L'overloading può essere fatto purché numero e/o tipo dei parametri sia diverso.
- Per non esagerare nel numero di parametri si può utilizzare il valore di default per uno o più parametri.

# Inizializzazione degli attributi

- Un'alternativa che ci aiuta a ridurre il numero di costruttori è la possibilità di inizializzare gli attributi della classe.

---

```
class Date{  
    int d {today.d};  
    int m {today.m};  
    int y {today.y};  
public:  
    Date(int, int, int);  
    // ...  
}
```

---

- Un costruttore può tuttavia inizializzare uno o più attributi:

---

```
Date::Date(int dd)  
    :d{dd} { // controlla se la data e' valida }
```

---

- Se mettiamo insieme i pezzi, dopo aver usato questo costruttore il valore di `d` è `dd`, mentre `m` e `y` valgono rispettivamente `today.m` e `today.y`.

# Definizione di metodi dentro la classe

- ▶ Due possibilità per i metodi:
  1. Dichiararli dentro definizione della classe e definirli fuori.
  2. Dichiararli e definirli dentro la definizione della classe.
- ▶ Preferibile il primo, salvo che il metodo sia piccolo, modificato raramente e usato spesso: infatti viene considerato come `inline`
- ▶ Pensate che la definizione della classe viene inclusa negli header, e quindi finisce in un certo numero di unità di traduzione.
- ▶ Ogni membro della classe può accedere a qualsiasi altro membro della classe indipendentemente da dove è stato definito: in altre parole, dichiarazione e definizione dei membri di una classe non dipendono dall'ordine.

# Esempio

Sono del tutto equivalenti

---

```
class Date{
public:
    void add_month(int n) {m+=n;}
    // ...
private:
    int d,m,y;
};
```

---

---

```
class Date{
public:
    void add_month(int n);
    // ...
private:
    int d,m,y;
};

inline void Date::add_month(int n) {
    m+=n;
}
```

---

# const

- ▶ Si può aggiungere ai metodi la parola chiave per sottolineare che si tratta di metodi che non vanno a modificare lo stato dell'oggetto.
- ▶ Si permette così al compilatore di controllare.

---

```
class Date{
    int d,m,y;
public:
    int day() const {return d;}
    int month() const {return m;}
    int year() const;
    // ...
};

void Date::year() const {
    return y;
}
```

---

- ▶ La parola chiave `const` è obbligatoria anche quando il metodo viene definito fuori della classe: fa parte della **signature** del metodo.

# Oggetto `const`

- ▶ Se l'oggetto è costante, non permette l'invocazione di metodi non dichiarati come costanti!
- ▶ Un metodo costante, ovviamente, può venire invocato anche su oggetti non costanti.

# Membri `static`

- ▶ I membri dichiarati `static` sono collegati alla **classe**, e non alle singole istanze.
- ▶ Vale sia per gli attributi, che per i metodi (che, ad esempio, vanno a modificare gli attributi statici).
- ▶ **Esempio**: valori di default per gli attributi che valgono per tutti i membri della classe e un metodo che va a impostare i valori di default.

# Costruttori e distruttori

- ▶ In C++ il nome del costruttore è `NomeClasse`; tra i compiti dei costruttori c'è quello di imporre degli **invarianti di classe**: proprietà che devono valere per ogni istanza di una classe.
- ▶ Se il costruttore trova che un invariante non viene rispettato, di solito lancia una (o più) eccezioni e non procede con la costruzione.
- ▶ È importante controllare che tutte le risorse vengano rilasciate.
- ▶ Questo è tra i compiti del distruttore `NomeClasse`.
- ▶ Il distruttore viene chiamato  
implicitamente quando l'oggetto esce dal suo ambito di definizione  
esplicitamente con `delete`



# Classi e sottoclassi

- Per indicare che `Employee` è una sovraclassa di `Manager`:

---

```
class Manager : public Employee{  
    // ...  
}
```

---

- Costruzione bottom-up (dalla classe base verso le derivate) e distruzione top-down (in senso inverso).
- I costruttori non vengono automaticamente ereditati (devono cambiare per forza di cose!)

# virtual

- ▶ Metodi in una classe che possono venir ridefinite nelle sottoclassi: quale metodo viene chiamato dipende dal tipo dell'oggetto, non della variabile come succede senza `virtual`.
- ▶ Per default, tutti i metodi che ridefiniscono un metodo `virtual` sono `virtual`.
- ▶ Il tipo di ritorno può essere una sottoclasse rispetto al metodo definito nella sovraclasses.
- ▶ Aggiungere la pseudo inizializzazione `=0` per un metodo **pure virtual** (non ha implementazione e deve essere quindi ridefinito).
- ▶ Una classe con una o più metodi virtuali puri si dice **astratta**: non si possono creare oggetti di una classe astratta.
- ▶ Se invece si aggiunge `final` dopo la dichiarazione, quel metodo non può più venir ridefinito.

# Controllo degli accessi

- I membri di una classe possono essere:

`private`

`protected` Di solito per gli attributi si evita il livello `protected`, che viola l'incapsulamento.

`public`

- Anche la sovraclasses può avere analoghi livelli di accesso:

`private` Tutti i membri della sovraclasses sono privati nella sottoclasses.

`protected` Tutti i membri pubblici e protetti della sovraclasses diventano protetti nella sottoclasses.

`public` Tutti i membri della sovraclasses diventano privati.

# Templates

- ▶ Un tipo o un valore diventa un parametro nella definizione di una classe, una funzione o un alias di tipo.



