

Object-oriented programming (Programmazione ad oggetti) Parte 2

Liceo G.B. Brocchi
Classi terze Scientifico - opzione scienze applicate
Bassano del Grappa, Febbraio 2023
Prof. Giovanni Mazzocchin

Copiare gli oggetti

- In C++, di default, copiare un oggetto **b** in un oggetto **a** significa copiare i membri di b nei membri di a
- Possiamo costruire un oggetto che è copia di un altro oggetto già esistente
- Il metodo che si occupa di costruire un oggetto *a* di tipo *T* come copia di un altro oggetto *b* di tipo *T* si chiama costruttore di copia (**copy constructor**)



Copiare gli oggetti

```
class Date {  
private:  
    int month;  
    int day;  
    int year;  
public:  
    Date(int m, int d, int y): month(m), day(d), year(y) {}  
  
    int get_month() const {  
        return month;  
    }  
    int get_day() const {  
        return day;  
    }  
    int get_year() const {  
        return year;  
    }  
};
```

```
int main() {  
    Date d1(4, 12, 2021);  
    Date d2(d1);  
}
```

dopo le 2 istruzioni del main, in memoria la situazione è la seguente:

d1

month: 4
day: 12
year: 2021

d2

month: 4
day: 12
year: 2021

Copiare gli oggetti

```
class Date {  
private:  
    int month;  
    int day;  
    int year;  
public:  
    Date(int m, int d, int y): month(m), day(d), year(y) {}  
  
    int get_month() const {  
        return month;  
    }  
    int get_day() const {  
        return day;  
    }  
    int get_year() const {  
        return year;  
    }  
};
```

```
int main() {  
    Date d1(4, 12, 2021);  
    Date d2(d1);  
}
```

Evidentemente, qualcuno si è occupato di costruire d2 copiandoci dentro membro a membro d1

d1

month: 4
day: 12
year: 2021

d2

month: 4
day: 12
year: 2021

Il *copy constructor*

- Proviamo a capire chi è questo «qualcuno» che si è occupato di copiare *d1* in *d2*
- Innanzitutto è un costruttore, in quanto ha costruito l'oggetto *d2*
- È vero che è un costruttore, ma è sicuramente diverso dal costruttore a 3 parametri definito nel codice, in quanto accetta un oggetto di tipo *Date*. Provate a pensare a quale potrebbe essere il prototipo del costruttore di copia

//3-parameter constructor

```
Date(int m, int d, int y): month(m), day(d), year(y) {}
```


//call to copy constructor

```
Date d2(d1);
```

Il copy constructor

- Il copy constructor con il comportamento standard (copia membro a membro) è già presente
- Noi però lo ridefiniamo per capire quando viene chiamato

```
Date(const Date& d) {  
    this->month = d.month;  
    this->day = d.day;  
    this->year = d.year;  
    cout << "called Date copy constructor" << endl;  
}
```

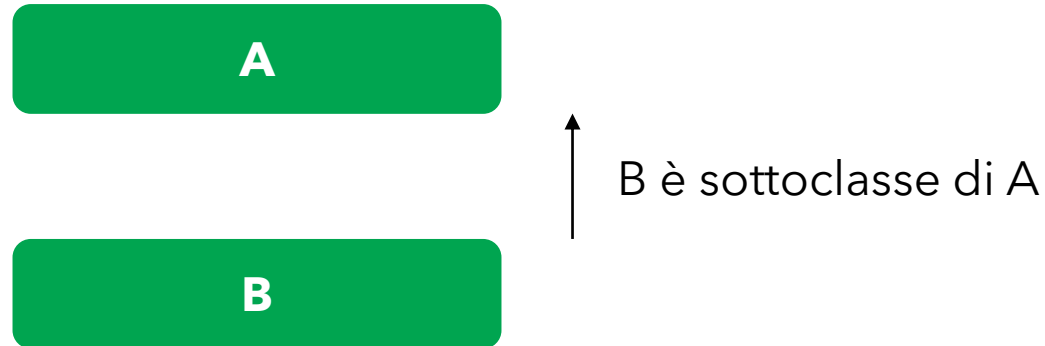


Ecco il prototipo del costruttore di copia:
T(const T&)
condivide il nome con la classe proprio perché
è un costruttore

in questo modo, quando viene creato un oggetto di tipo Date di copia da un altro oggetto di tipo Date, verrà stampato *called Date copy constructor*

Ereditarietà

- L'ereditarietà (*inheritance*) è una caratteristica della programmazione ad oggetti che consente di creare una classe **B** a partire da una classe **A** già esistente
- Si dice che la classe **B** *deriva* da **A**, o che **A** è una *base* di **B**, oppure si può dire che **B** è sottoclasse (*subclass*) di **A**, e **A** è **superclasse** (*superclass*) di **B**
- Con l'ereditarietà, la sottoclasse *eredita* tutti i membri della classe base



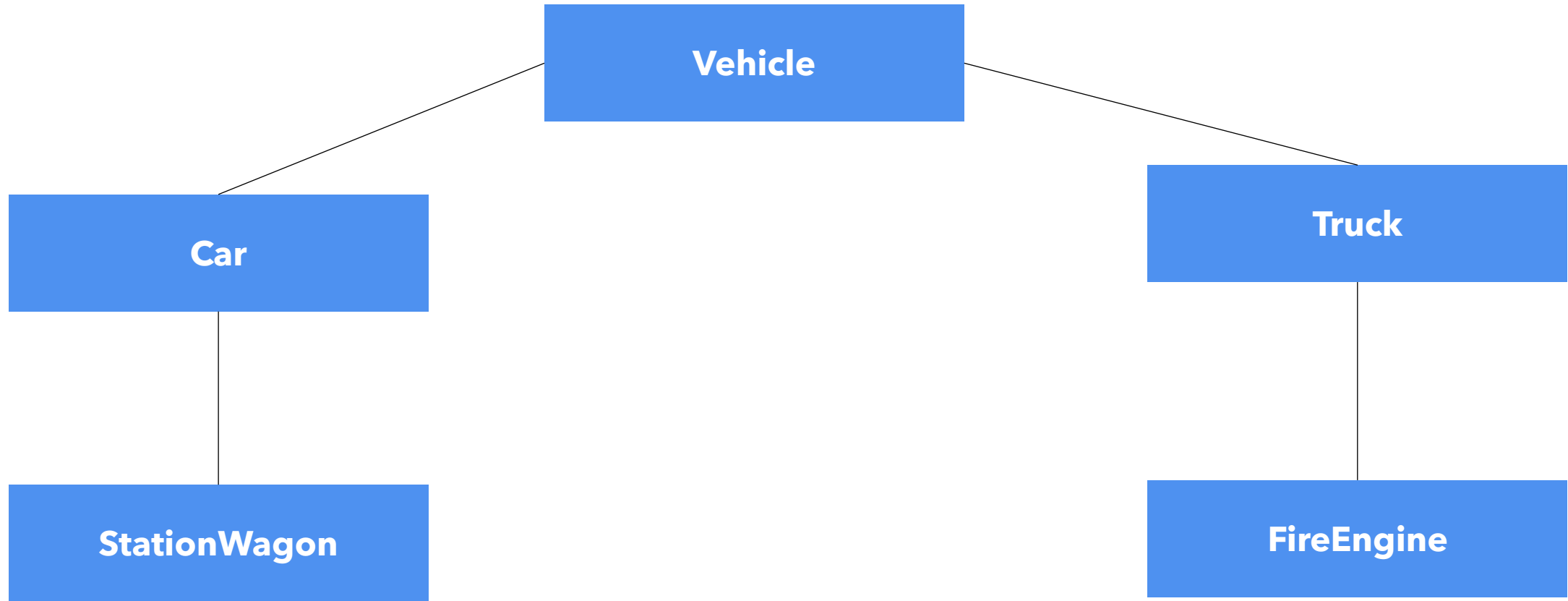
Ereditarietà

- Un esempio realistico: in un software per la gestione della produzione di veicoli, potrebbero esistere una classe `Vehicle` (veicolo generico), una classe `Car` (automobile generica) una classe `Truck` (camion), una classe `StationWagon` (automobile familiare), una classe `FireEngine` (camion dei vigili del fuoco), una classe `SportCar`
- Le classi elencate sopra non saranno scollegate. Infatti hanno senso le seguenti affermazioni:
 - una `Car` è un `Vehicle`
 - un `Truck` è un `Vehicle`
 - un `StationWagon` è una `Car`
 - un `FireEngine` è uno `Truck`

Ereditarietà

- Un esempio realistico: in un videogioco, potrebbero esistere una classe `Player` (giocatore generico), una classe `Weapon` (arma generica), una classe `Setting` (ambiente di gioco), una classe `PlayerWithSuperPowers` (giocatore «potenziato»), una classe `SettingLevel1` (ambiente di gioco per il livello 1), una classe `NuclearWeapon` etc...
- Sicuramente le classi elencate sopra non saranno scollegate. Infatti, hanno senso le seguenti affermazioni:
 - un `PlayerWithSuperPowers` è un `Player`
 - un `SettingLevel1` è un `Setting`
 - una `NuclearWeapon` è una `Weapon`

Una gerarchia di classi (*class hierarchy*)



<https://github.com/Cyofanni/high-school-cs-class/tree/main/cplusplus/oop/inheritance/vehicles>