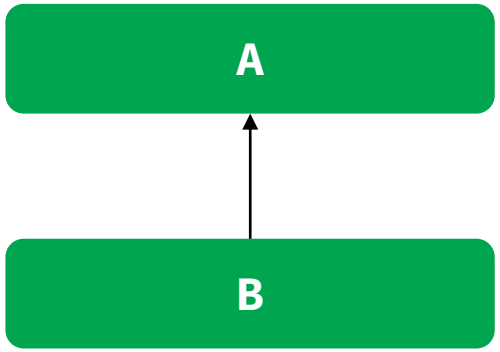
Object-oriented programming (Programmazione ad oggetti) Parte 3

Liceo G.B. Brocchi – Bassano del Grappa (VI)
Liceo Scientifico – opzione scienze applicate
Giovanni Mazzocchin

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**
- La sottoclasse *eredita* tutti i membri della classe base



B è sottoclasse di A

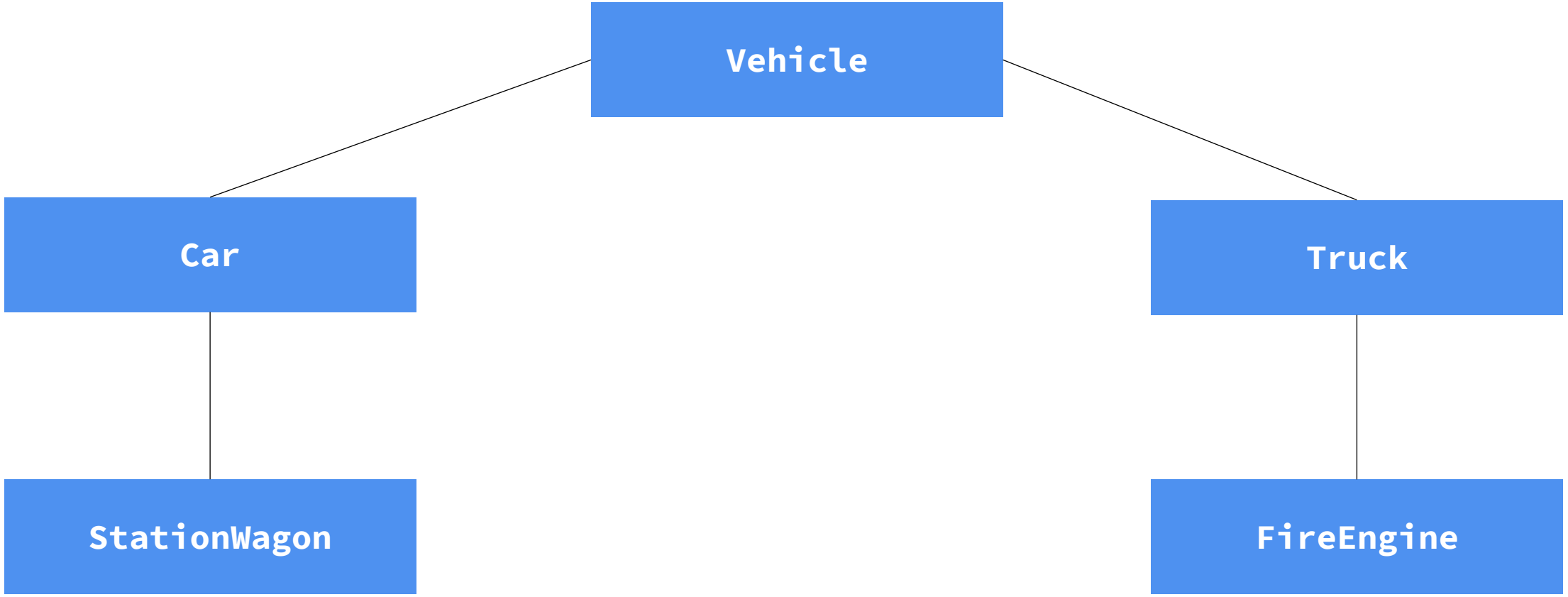
Ereditarietà

- **Esempio**: 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`
    - una `StationWagon` è una `Car`
    - un `FireEngine` è un `Truck`

Ereditarietà

- **Esempio**: in un videogioco, potrebbero esistere una classe `Player` (giocatore generico), una classe `Weapon` (arma generica), una classe `Setting` (ambiente di gioco), una classe `PlayerPlus` (giocatore potenziato), una classe `SettingLevel1` (ambiente di gioco per il livello 1), una classe `SuperWeapon` etc...
- Le classi elencate sopra non saranno scollegate. Infatti, hanno senso le seguenti affermazioni:
    - un `PlayerPlus` è un `Player`
    - un `SettingLevel1` è un `Setting`
    - una `SuperWeapon` è una `Weapon`

Ereditarietà: una *class hierarchy*



Ereditarietà

```
class Vehicle {
private:
  std::string serial_number;
public:
  Vehicle(std::string);
  std::string get_serial_number() const;
};

/*
  implementation…
*/
```

Ereditarietà

```
class Truck: public Vehicle {
private:
  double max_load_kg;
public:
  Truck(std::string, double);
  double get_max_load_kg() const;
};

Truck::Truck(std::string sn, double ml):
  Vehicle(sn), max_load_kg(ml) {}
```

costruzione del sottooggetto

Ereditarietà

```
int main(int argc, char* argv[]) {
  Vehicle v("BA564QW");
  Truck t("TA574TB", 1000);
  cout << t.get_serial_number() << endl;
}
```

Ereditarietà: membri `protected`

- Conosciamo già il significato dei modificatori di accesso `private` e `public`
- Un membro con modificatore di accesso `protected` in una classe `C` è visibile soltanto in `C` e nelle sottoclassi di `C`
- Modificare gli esempi precedenti aggiungendo un membro `protected` e sperimentare