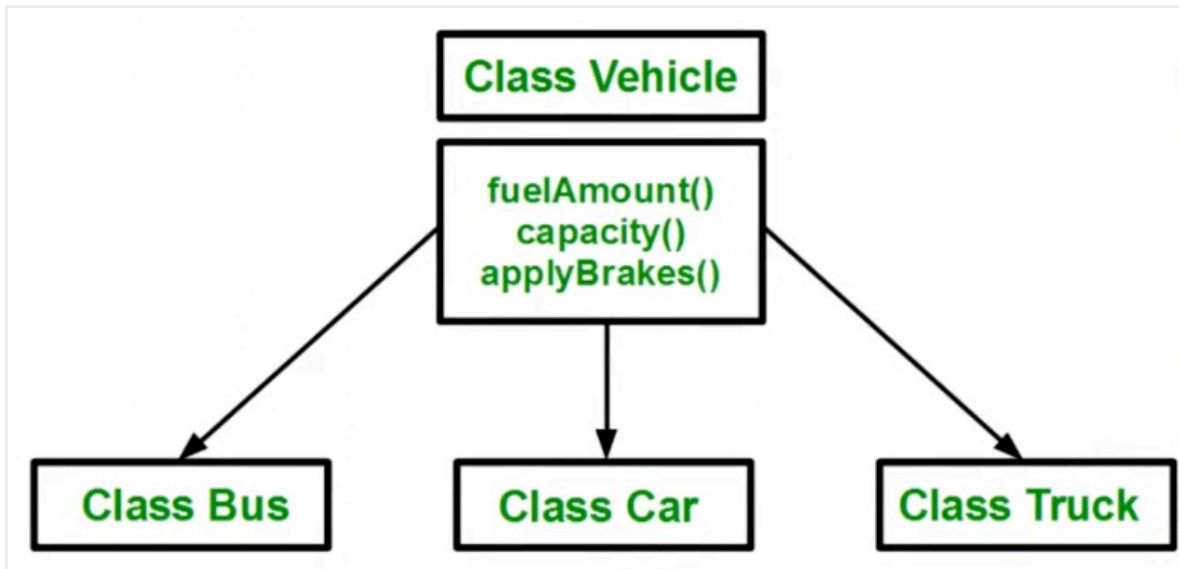


Inheritance

- Class and struct can inherit data and functions from other classes.
- There are three types of inheritance in C++:
 - public
 - protected
 - private
- Public inheritance keeps all access specifiers of the base class.



```
C++ inheritance_example.cpp inheritance_example.cpp/Vehicle
3   int capacity_ = 0; // amount of fuel of the gas tank
4   string brand_;    // make of the vehicle
5
6   public:
7   Vehicle(int seats, int capacity, string brand)
8       : seats_(seats), capacity_(capacity), brand_(std::move(brand)) {}
9
10  int get_seats() const { return seats_; }
11  int get_capacity() const { return capacity_; }
12  string get_brand() const { return brand_; }
13
14  void Print() const {
15      cout << "Vehicle seats available = " << seats_ << endl
16          << "Vehicle fuel available = " << capacity_ << endl
17          << "Vehicle brand = " << brand_ << endl;
18  }
19  };
20
21  //////////////////////////////////////////////////
22  class Bus : public Vehicle {
23  public:
24      Bus(int seats, int capacity, const string& brand, string brakes)
25          : Vehicle(seats, capacity, brand), special_brakes_(std::move(brakes)) {}
26
27      string special_brakes() const { return special_brakes_; }
28
29  private:
30      string special_brakes_;
31  };
32
33  //////////////////////////////////////////////////
34  class Car : public Vehicle {
35  public:
36      Car(int seats, int capacity, const string& brand, string stereo)
37          : Vehicle(seats, capacity, brand), stereo_brand_(std::move(stereo)) {}
38  }
```

```
C++ inheritance_example.cpp inheritance_example.cpp Car
16 };
15
14 //////////////////////////////////////////////////
13 class Bus : public Vehicle {
12 public:
11     Bus(int seats, int capacity, const string& brand, string brakes)
10         : Vehicle(seats, capacity, brand), special_brakes_(std::move(brakes)) {}
9
8     string special_brakes() const { return special_brakes_; }
7
6 private:
5     string special_brakes_;
4 };
3
2 //////////////////////////////////////////////////
1 class Car : public Vehicle {
49 public:
1     Car(int seats, int capacity, const string& brand, string stereo)
2         : Vehicle(seats, capacity, brand), stereo_brand_(std::move(stereo)) {}
3
4     string stereo_brand() const { return stereo_brand_; }
5
6 private:
7     string stereo_brand_;
8 };
9
10 //////////////////////////////////////////////////
11 int main() {
12     Bus my_bus{20, 100, "Volkswagen", "LPM_178"};
13     my_bus.Print();
14
15     Car my_car{4, 60, "Ford", "Sony"};
16     my_car.Print();
17     return 0;
18 }
19
```

Public Inheritance

- Public Inheritance stands for "is a" relationship, if class Derived inherits publicly from class Base, we say that, Derived is a kind of Base.
- Allows Derived to use all public and protected members of Base.
- Derived still gets its own special functions:
Constructors, Destructor, Assignment operators

```

1 #include <iostream>
2 using std::cout; using std::endl;
3 class Rectangle {
4     public:
5         Rectangle(int w, int h) : width_{w}, height_{h} {}
6         int width() const { return width_; }
7         int height() const { return height_; }
8     protected:
9         int width_ = 0;
10        int height_ = 0;
11 };
12 class Square : public Rectangle {
13     public:
14         explicit Square(int size) : Rectangle{size, size} {}
15 };
16 int main() {
17     Square sq(10); // Short name to save space.
18     cout << sq.width() << " " << sq.height() << endl;
19     return 0;
20 }

```

Function Overriding

- A function can be declared virtual
virtual Func(<params>);
- If function is virtual in base class it can be overridden in Derived class
Func(<params>) override;
- Base can force all Derived classes to override a function by making it pure virtual
virtual Func(<params>) = 0;

Overloading vs overriding

- Do not confuse function **overloading** and **overriding**
- **Overloading:**
 - Pick from all functions with the **same name**, but **different parameters**
 - Pick a function **at compile time**
 - Functions don't have to be in a class
- **Overriding:**
 - Pick from functions with the **same arguments and names** in different classes of **one class hierarchy**

Abstract classes and Interfaces

- Abstract class: class has at least one pure virtual function.
- Interface: class that has only pure virtual functions and no data members.

How virtual works

- A class with virtual functions has a virtual table.
- When calling a function the class checks which of the virtual function that match the signature should be called.
- Called runtime polymorphism.
- Costs some time but is very convenient.

Using Interfaces

- Use interfaces when you must enforce other classes to implement some functionality.
- Allow thinking about classes in terms of abstract functionality.
- Hide implementation from the caller.
- Allow to easily extend functionality by simply adding a new class.

```
1 #include <iostream>
2 using std::cout;
3 using std::endl;
4 struct Printable { // Saving space. Should be a class.
5     virtual void Print() const = 0;
6 };
7 struct A : public Printable {
8     void Print() const override { cout << "A" << endl; }
9 };
10 struct B : public Printable {
11     void Print() const override { cout << "B" << endl; }
12 };
13 void Print(const Printable& var) { var.Print(); }
14 int main() {
15     Print(A());
16     Print(B());
17     return 0;
18 }
```