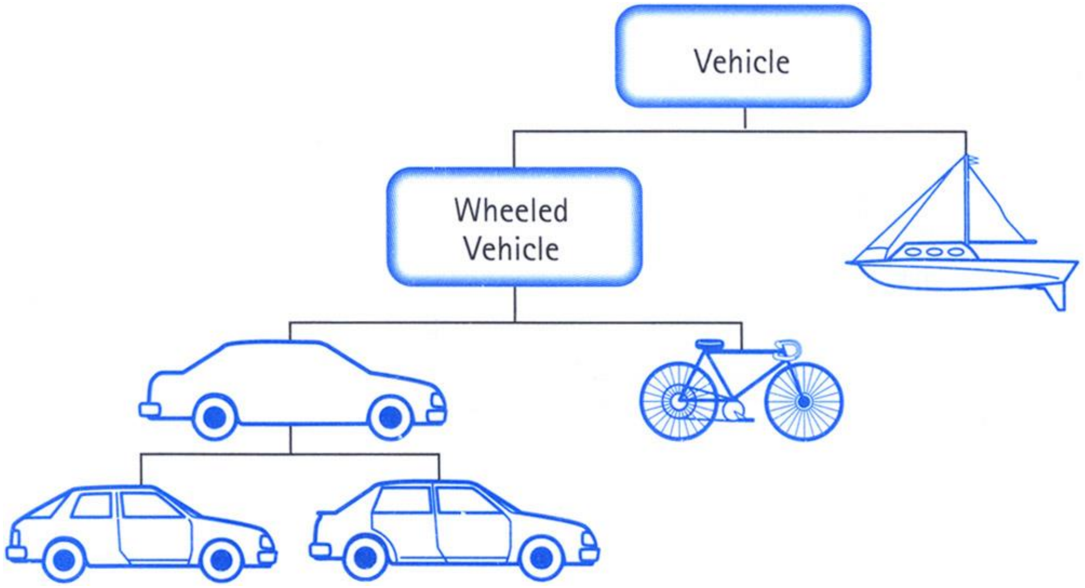
Techimax

Fundamentals of Programming in C++

Object Oriented Programming

In traditional programming, an object is a piece of memory to store values.

In object-oriented programming, an "object" implies that it is both an object in the traditional programming sense, and that it combines both properties and behaviors.

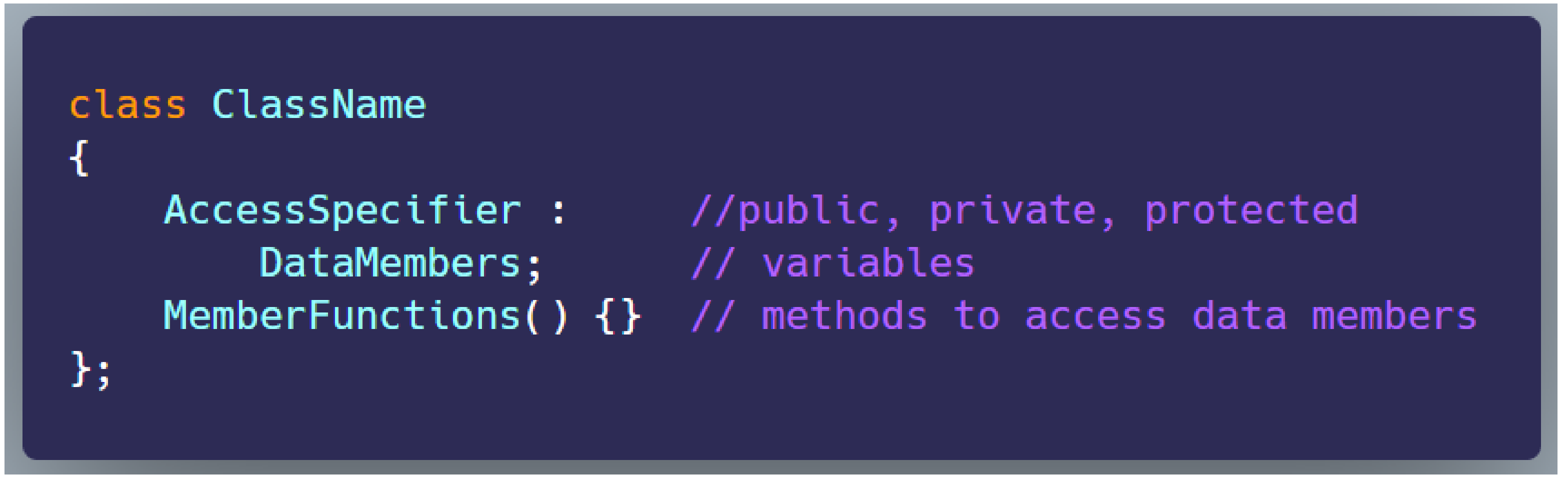


Object Oriented Programming

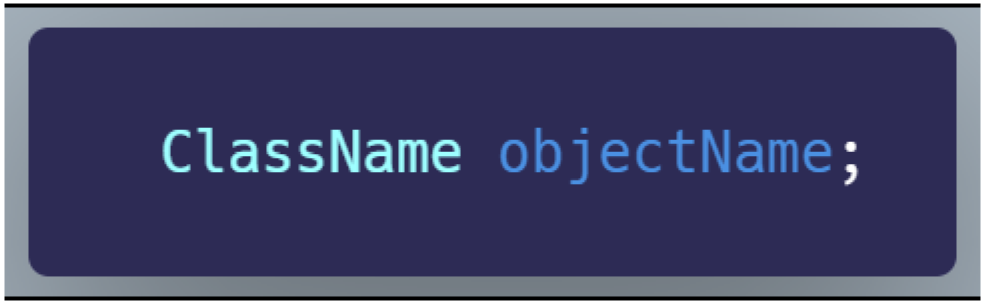
Classes	Objects
User defined data type, a blueprint from which an instance can be created.	Instance of a class
Bind data as well as methods together in a single unit.	Acts like a variable of the class
Have a logical existence	Have a physical existence
Does not take up any memory	Takes up memory.
To be declared only once.	Can be declared several times.

Object Oriented Programming

Syntax



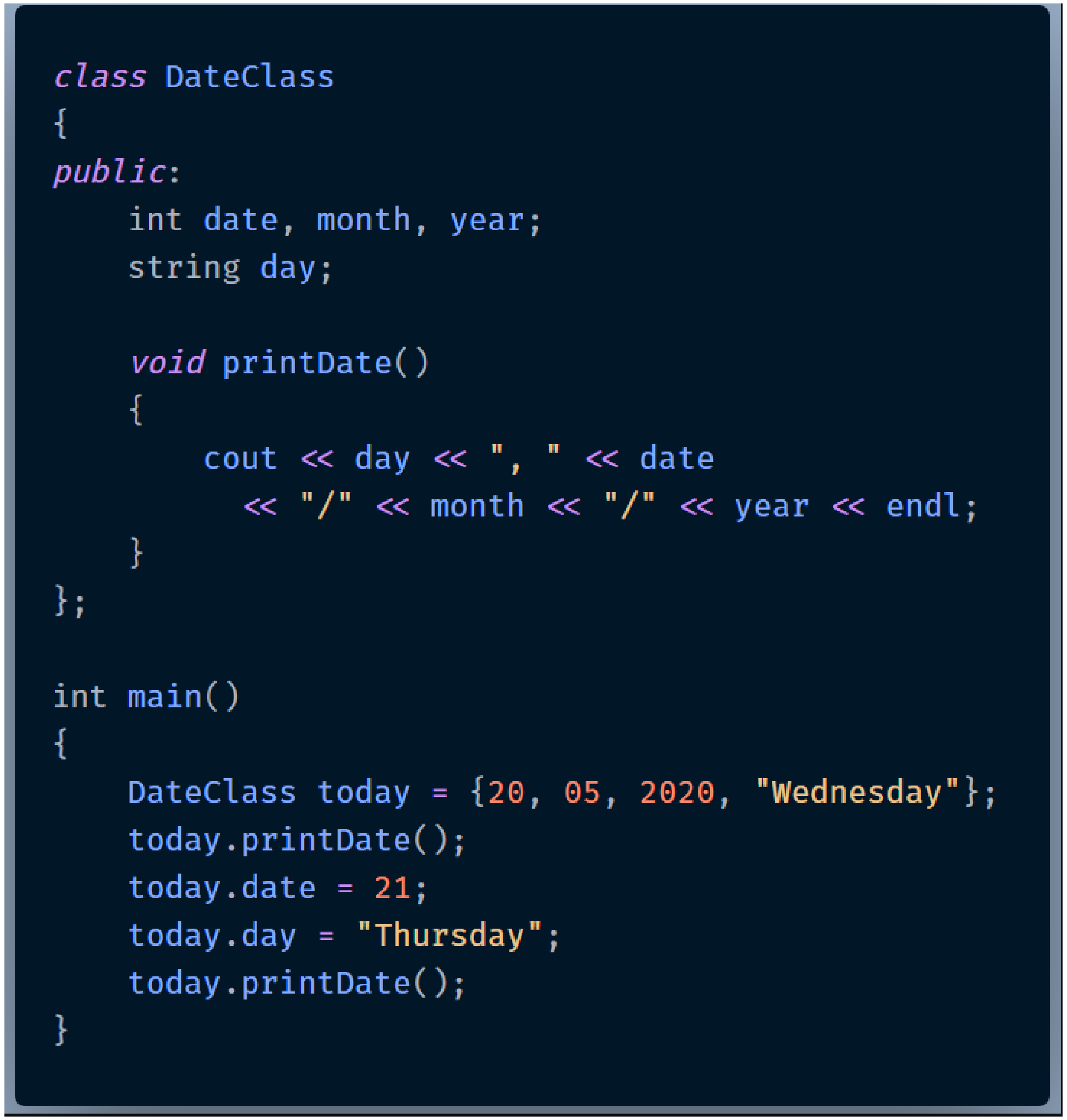
```
class ClassName
{
    AccessSpecifier :      //public, private, protected
        DataMembers;       // variables
    MemberFunctions() {}   // methods to access data members
};
```



```
ClassName objectName;
```

Object Oriented Programming

Using Classes & Objects



```
class DateClass
{
public:
    int date, month, year;
    string day;

    void printDate()
    {
        cout << day << ", " << date
          << "/" << month << "/" << year << endl;
    }
};

int main()
{
    DateClass today = {20, 05, 2020, "Wednesday"};
    today.printDate();
    today.date = 21;
    today.day = "Thursday";
    today.printDate();
}
```

Object Oriented Programming

Access Specifiers/Modifiers

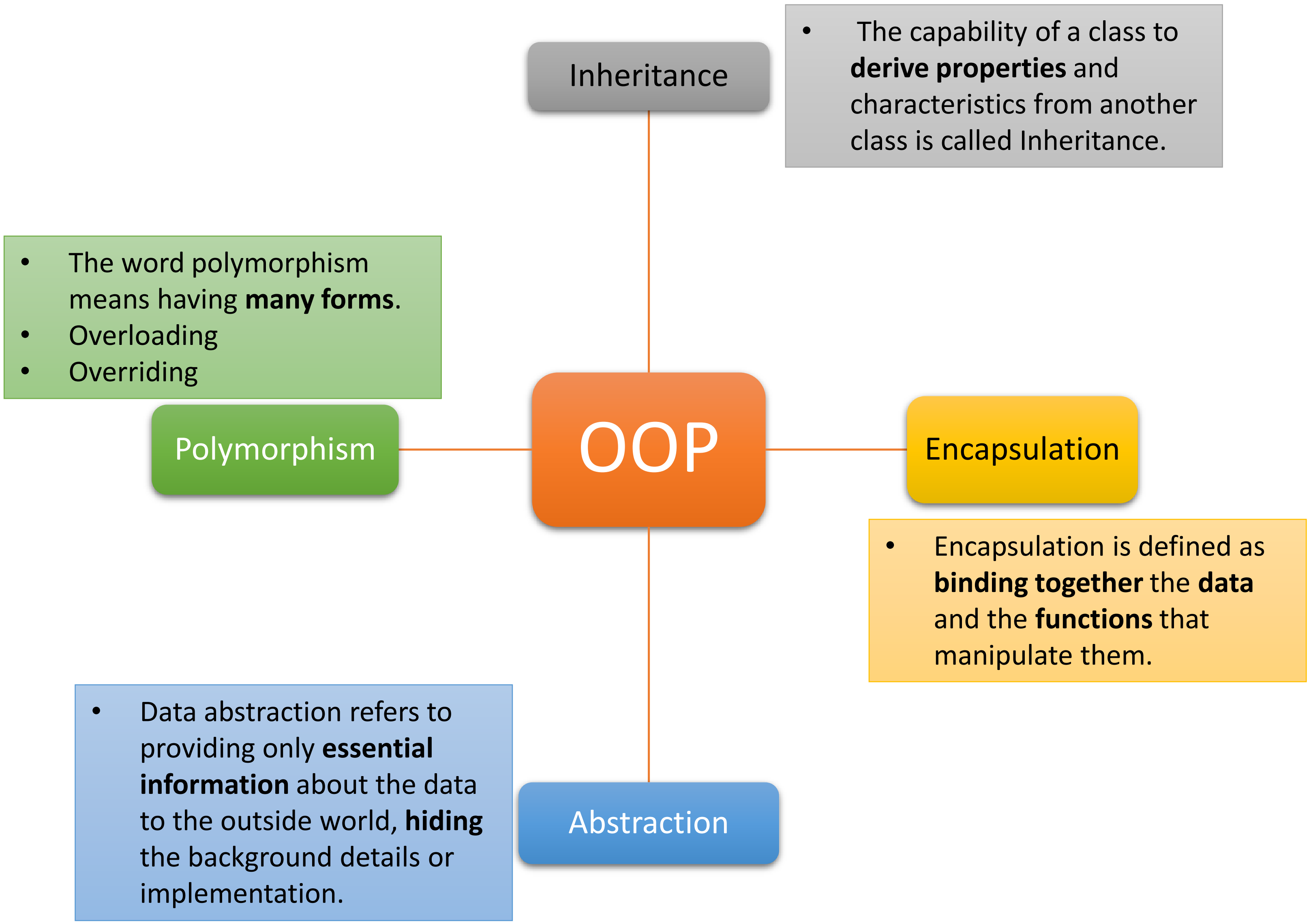
Public members are members of a struct or class that can be accessed from outside of the struct or class.

Private members are members of a class that can only be accessed by other members of the class.

The **protected** access specifier allows the class the member belongs to, friends, and derived classes to access the member. However, protected members are not accessible from outside the class.

Access Modifier	Own Class	Derived Class	Outside the Class
Public	Yes	Yes	Yes
Private	Yes	No	No
Protected	Yes	Yes	No

Object Oriented Programming



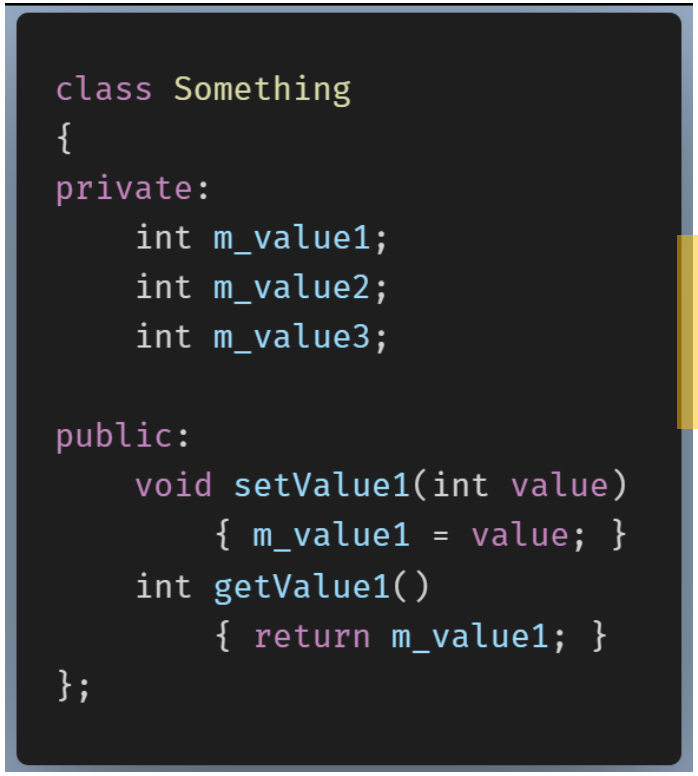
- The capability of a class to **derive properties** and characteristics from another class is called Inheritance.

- The word polymorphism means having **many forms**.
- Overloading
- Overriding

- Encapsulation is defined as **binding together** the **data** and the **functions** that manipulate them.

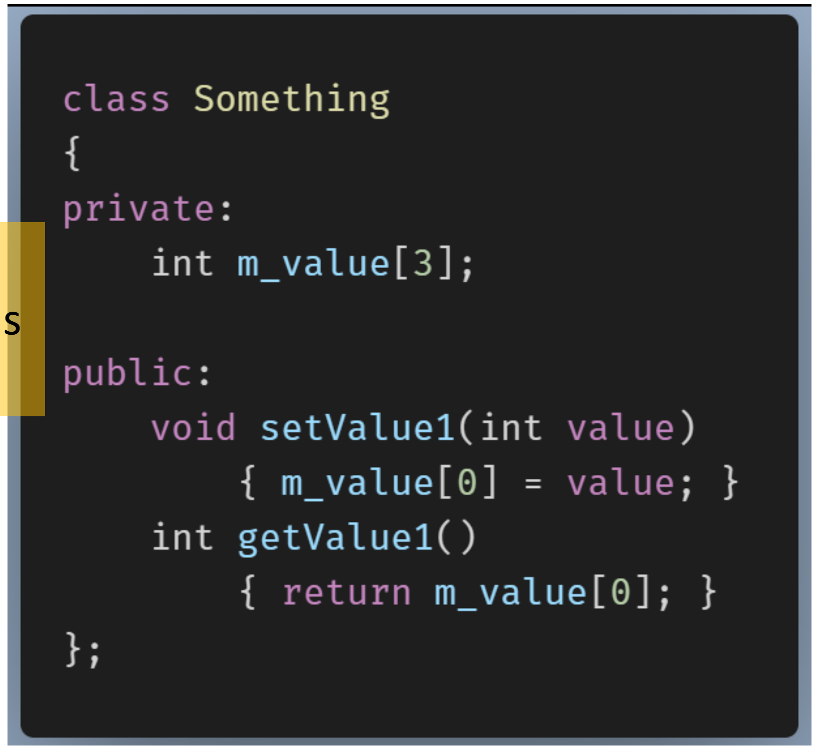
- Data abstraction refers to providing only **essential information** about the data to the outside world, **hiding** the background details or implementation.

Abstraction and Encapsulation

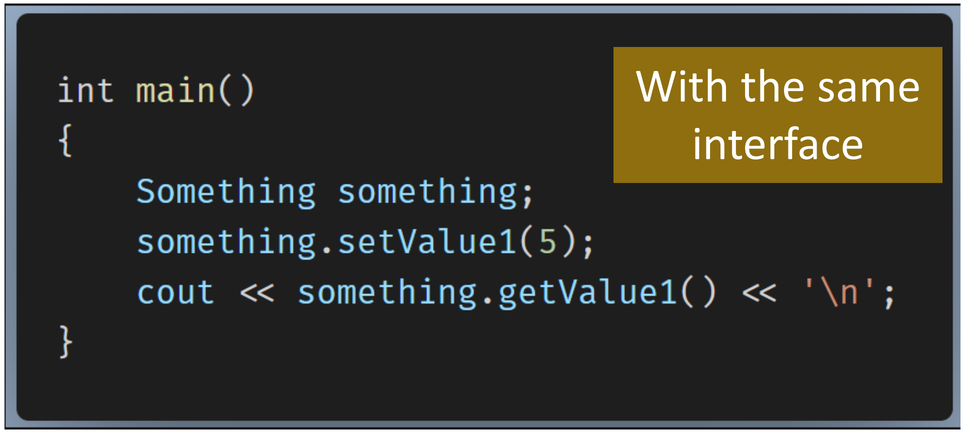


```cpp
class Something
{
private:
    int m_value1;
    int m_value2;
    int m_value3;

public:
    void setValue1(int value)
        { m_value1 = value; }
    int getValue1()
        { return m_value1; }
};
```

Both the implementations work



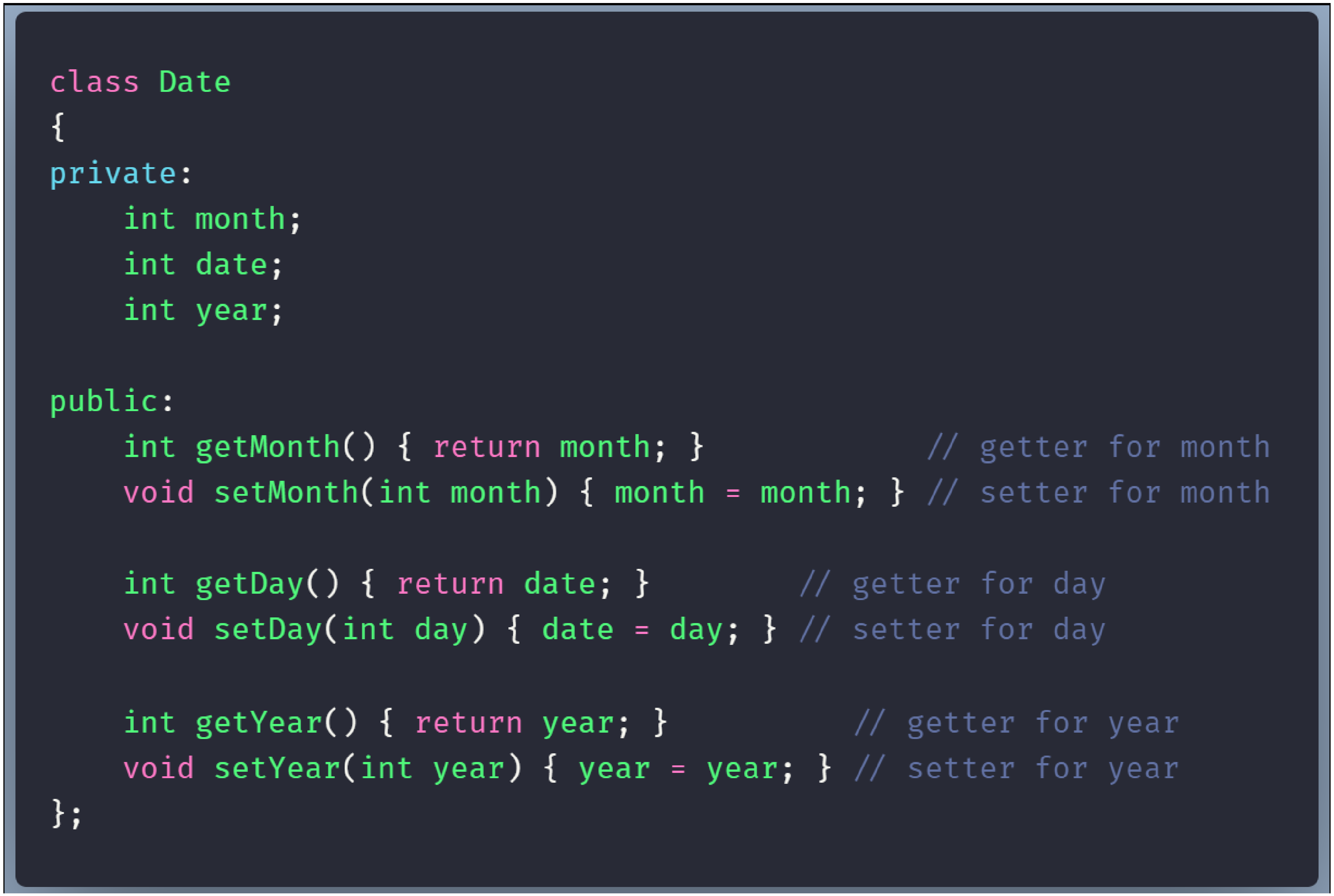
```cpp
class Something
{
private:
    int m_value[3];

public:
    void setValue1(int value)
        { m_value[0] = value; }
    int getValue1()
        { return m_value[0]; }
};
```



```cpp
int main()
{
    Something something;
    something.setValue1(5);
    cout << something.getValue1() << '\n';
}
```

With the same interface

Getters and Setters



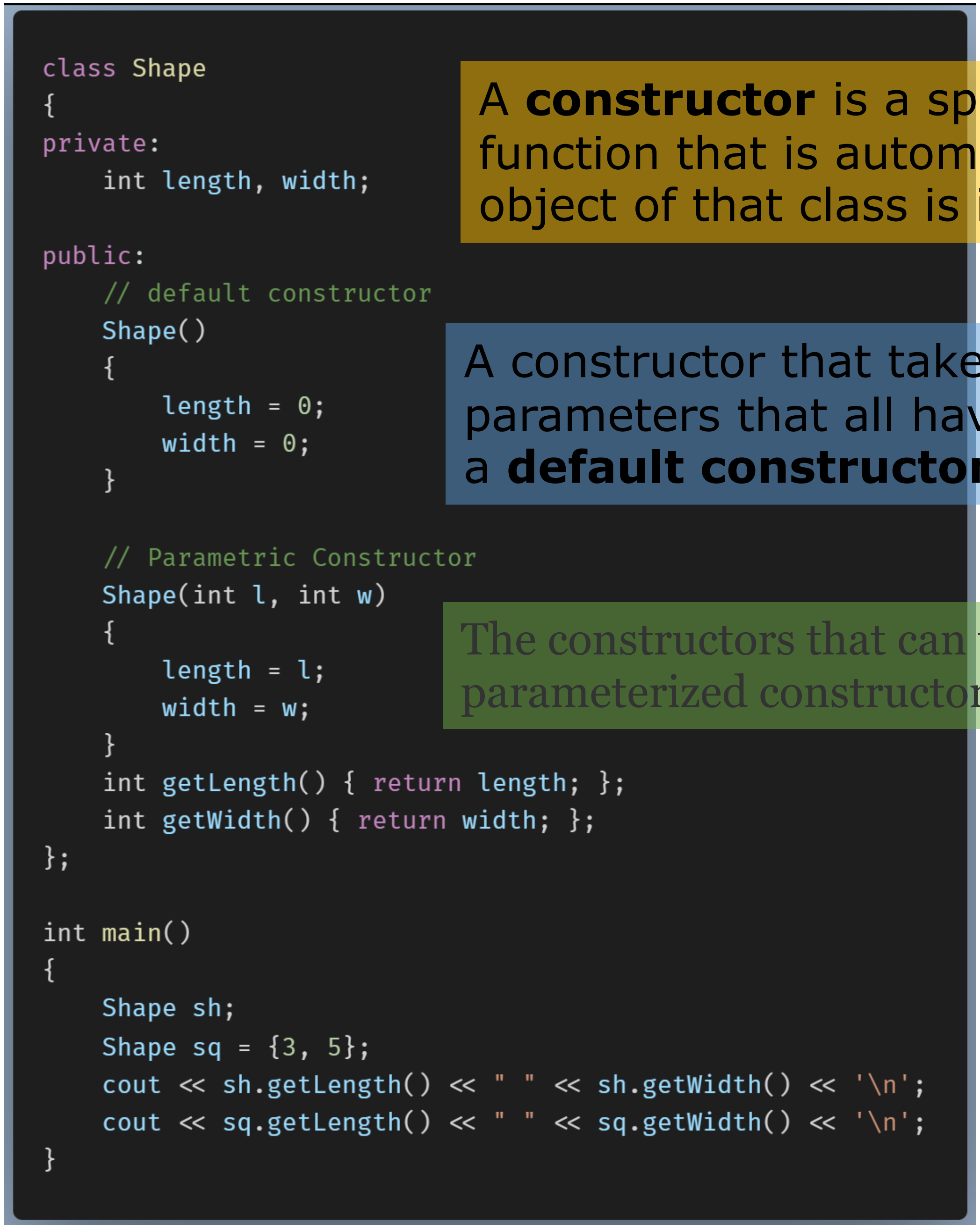
```
class Date
{
private:
    int month;
    int date;
    int year;

public:
    int getMonth() { return month; }            // getter for month
    void setMonth(int month) { month = month; } // setter for month

    int getDay() { return date; }        // getter for day
    void setDay(int day) { date = day; } // setter for day

    int getYear() { return year; }          // getter for year
    void setYear(int year) { year = year; } // setter for year
};
```

Constructors

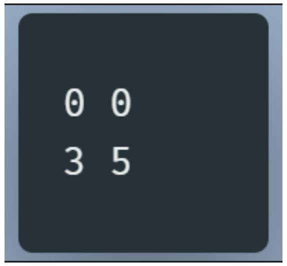
A **constructor** is a special kind of class member function that is automatically called when an object of that class is instantiated.

A constructor that takes no parameters (or has parameters that all have default values) is called a **default constructor**.

The constructors that can take arguments are called parameterized constructors.



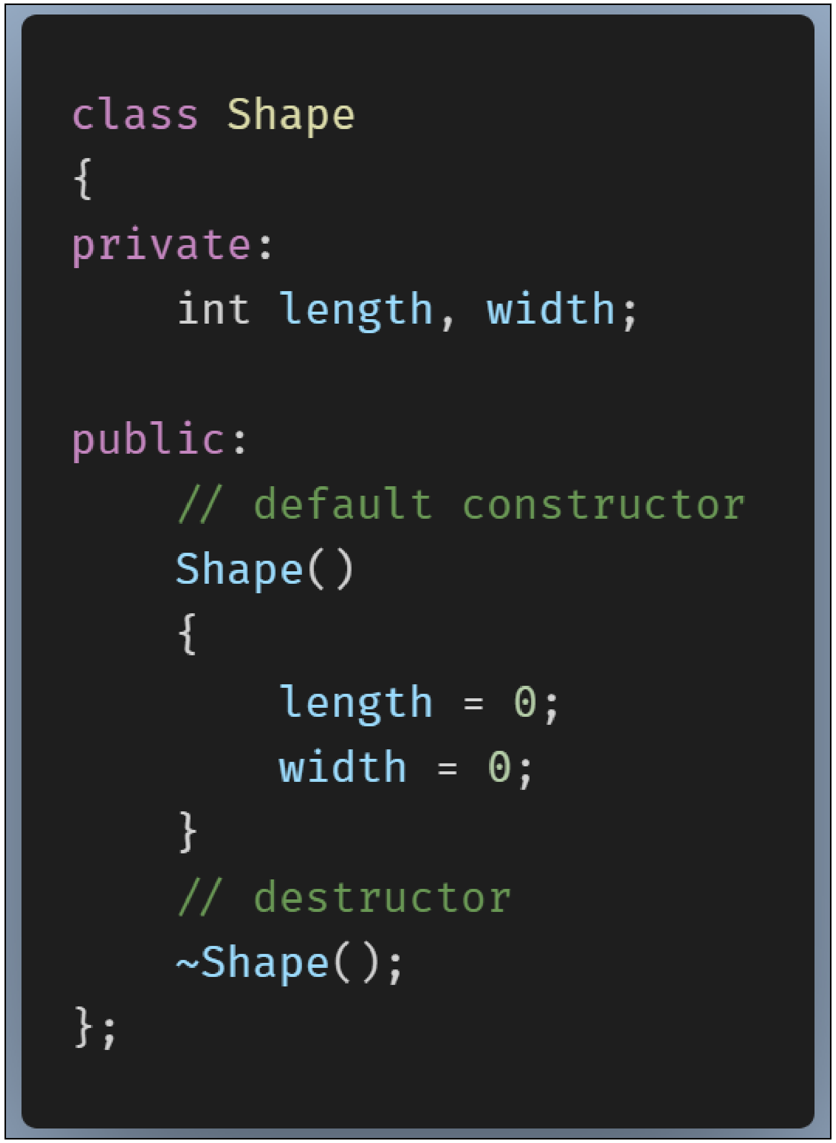
```cpp
class Shape
{
private:
    int length, width;

public:
    // default constructor
    Shape()
    {
        length = 0;
        width = 0;
    }

    // Parametric Constructor
    Shape(int l, int w)
    {
        length = l;
        width = w;
    }
    int getLength() { return length; };
    int getWidth() { return width; };
};

int main()
{
    Shape sh;
    Shape sq = {3, 5};
    cout << sh.getLength() << " " << sh.getWidth() << '\n';
    cout << sq.getLength() << " " << sq.getWidth() << '\n';
}
```



```
0 0
3 5
```

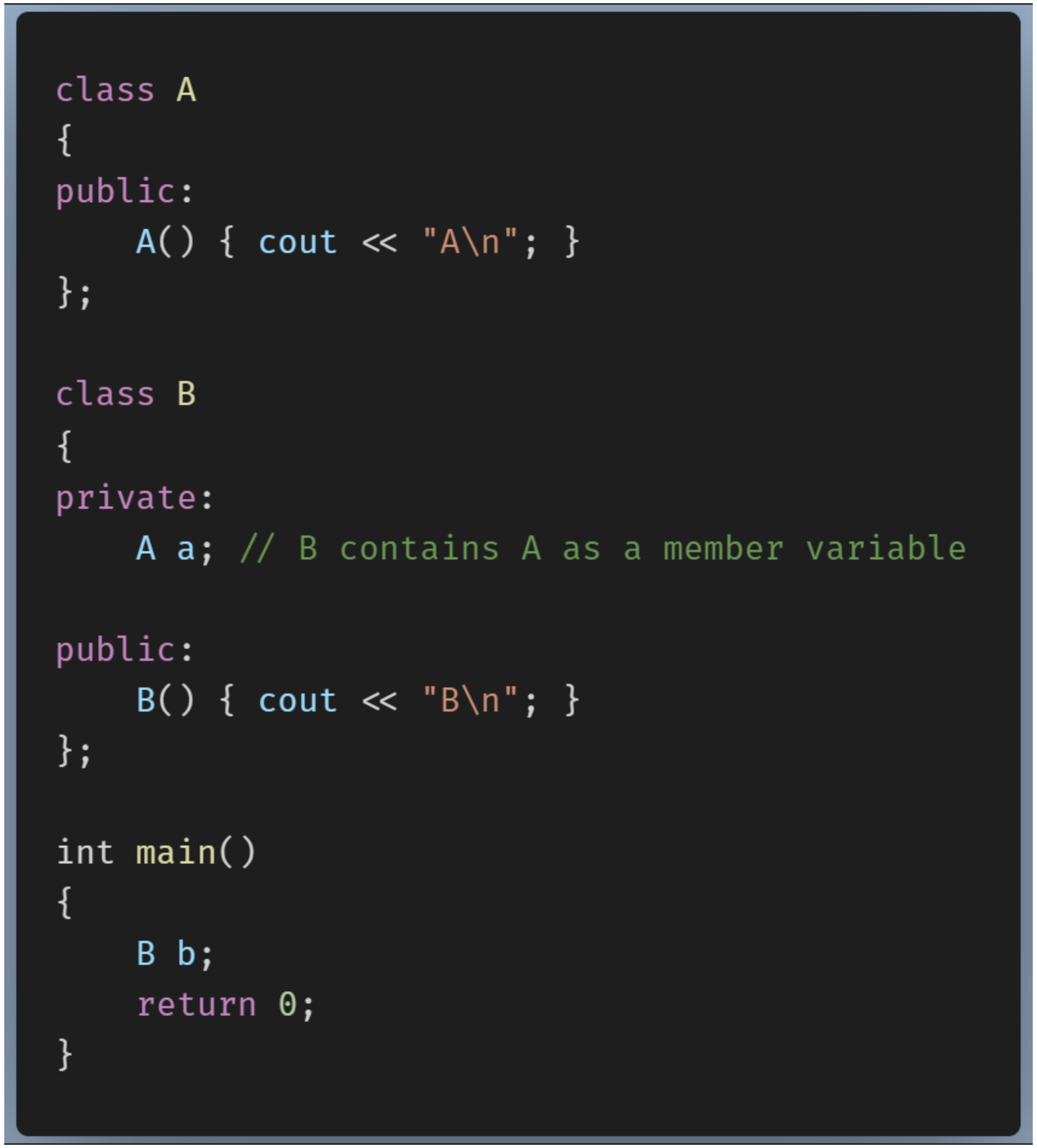
Destructors

Destructor is another special member function that is called by the compiler when the scope of the object ends.

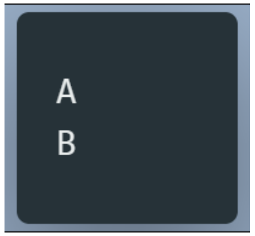


```
class Shape
{
private:
    int length, width;

public:
    // default constructor
    Shape()
    {
        length = 0;
        width = 0;
    }
    // destructor
    ~Shape();
};
```

Class within a Class



```
class A
{
public:
    A() { cout << "A\n"; }
};

class B
{
private:
    A a; // B contains A as a member variable

public:
    B() { cout << "B\n"; }
};

int main()
{
    B b;
    return 0;
}
```

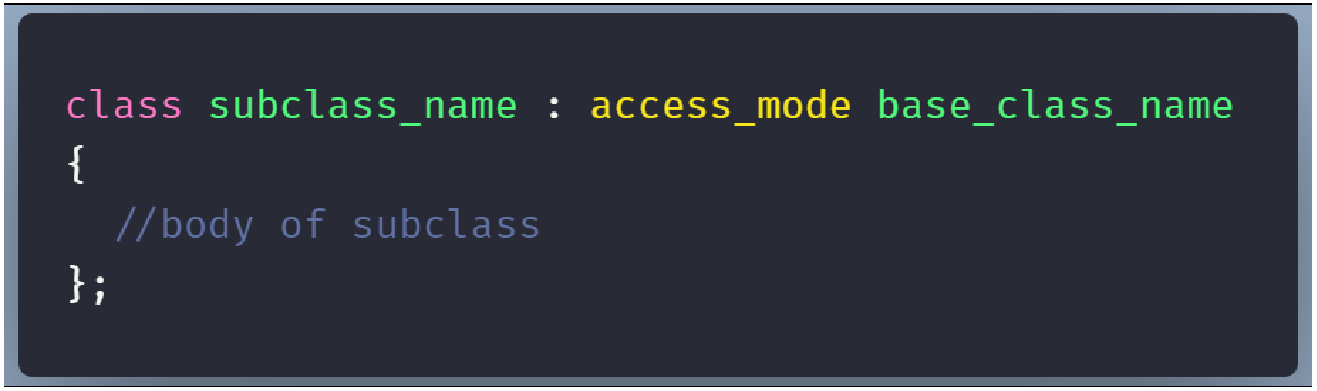


```
A
B
```

Inheritance

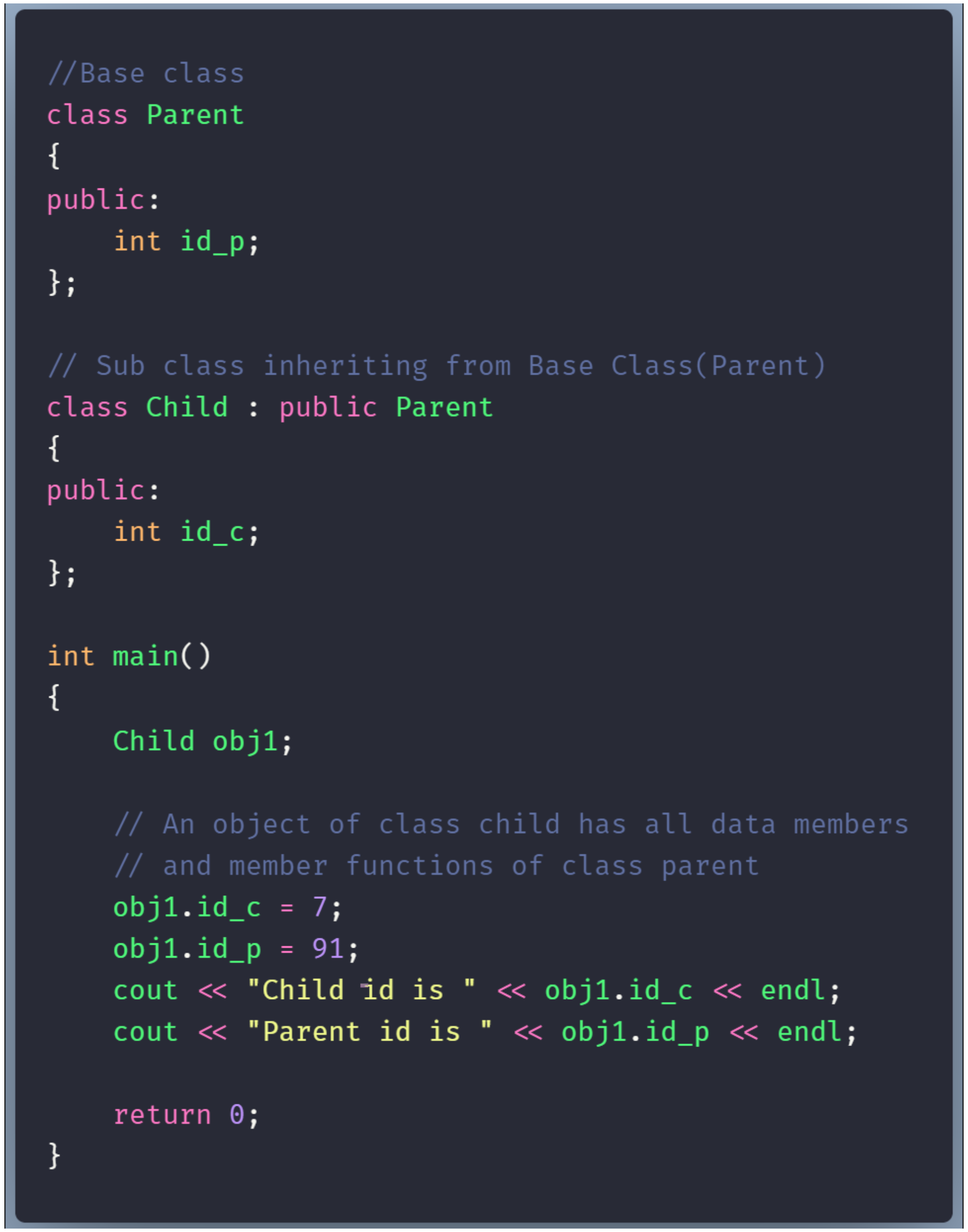
Super Class
The class whose properties are inherited by sub class is called Base Class or Super class.

Sub Class
The class that inherits properties from another class is called Sub class or Derived Class.



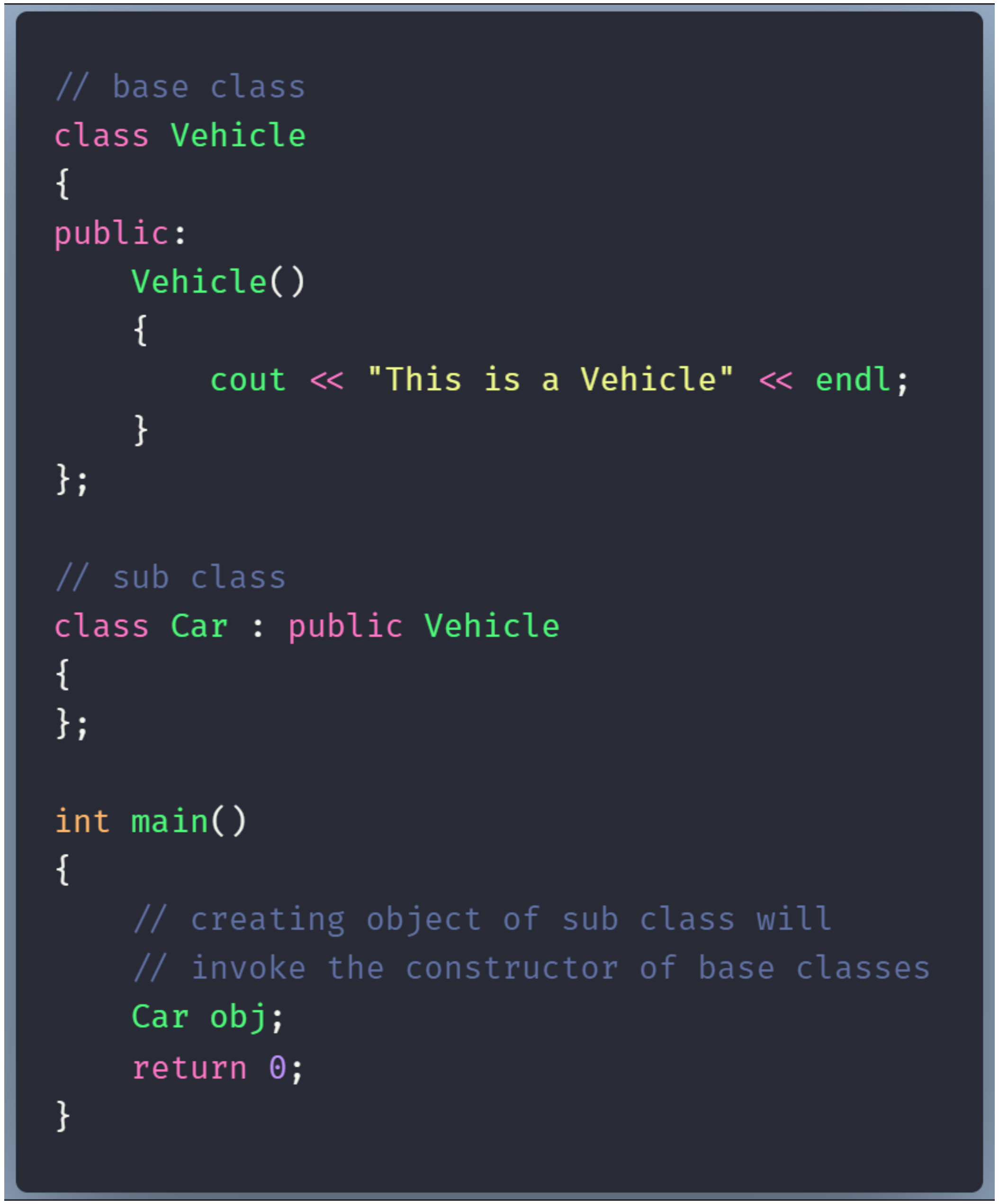
```
class subclass_name : access_mode base_class_name
{
  //body of subclass
};
```

Inheritance

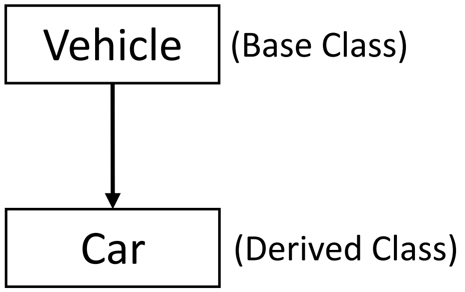


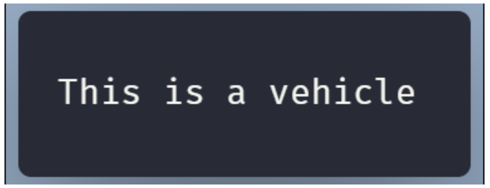
```
//Base class
class Parent
{
public:
    int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
public:
    int id_c;
};

int main()
{
    Child obj1;

    // An object of class child has all data members
    // and member functions of class parent
    obj1.id_c = 7;
    obj1.id_p = 91;
    cout << "Child id is " << obj1.id_c << endl;
    cout << "Parent id is " << obj1.id_p << endl;

    return 0;
}
```

Single Inheritance



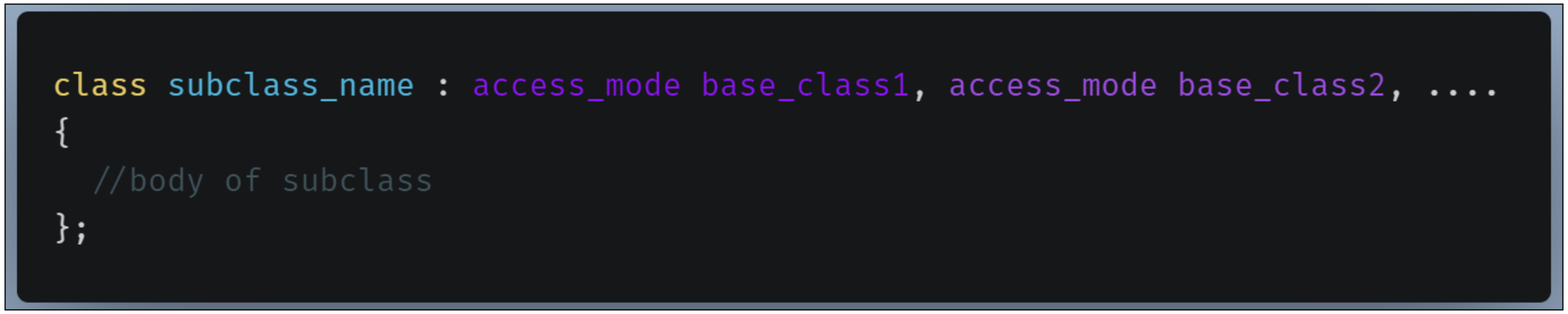
```
// base class
class Vehicle
{
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};

// sub class
class Car : public Vehicle
{
};

int main()
{
    // creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```



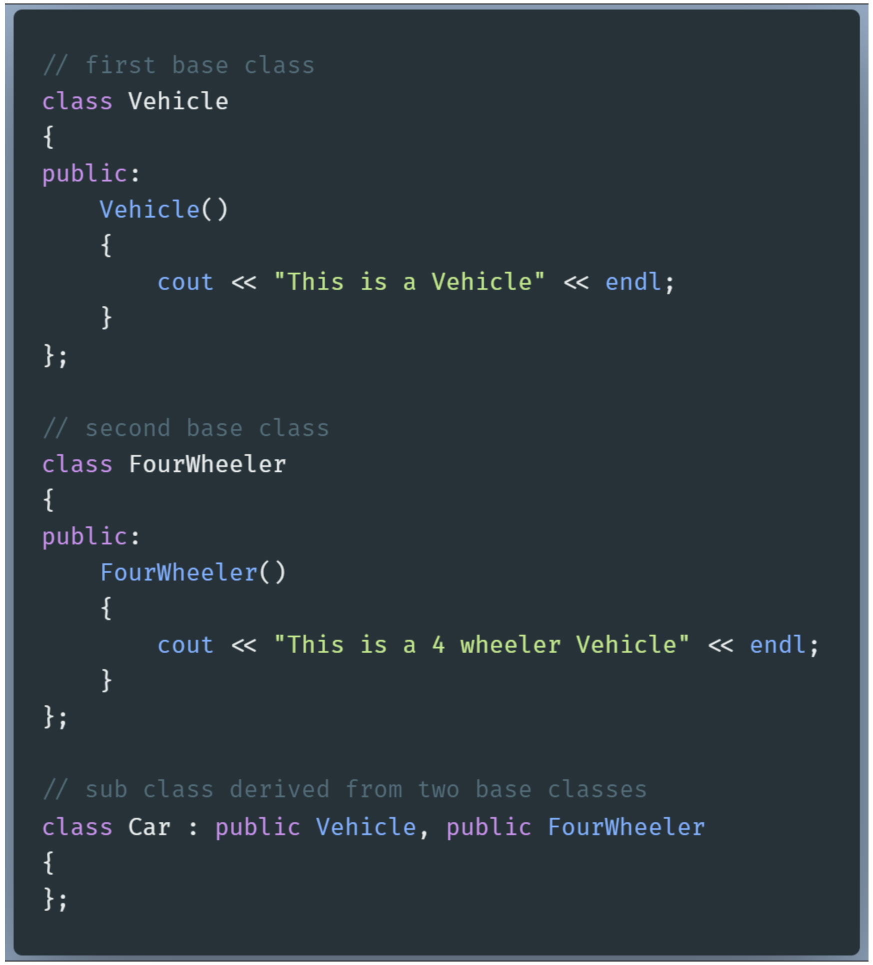


```
This is a vehicle
```

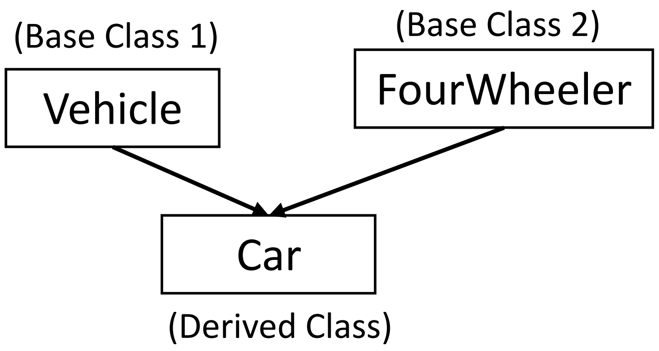
Multiple Inheritance



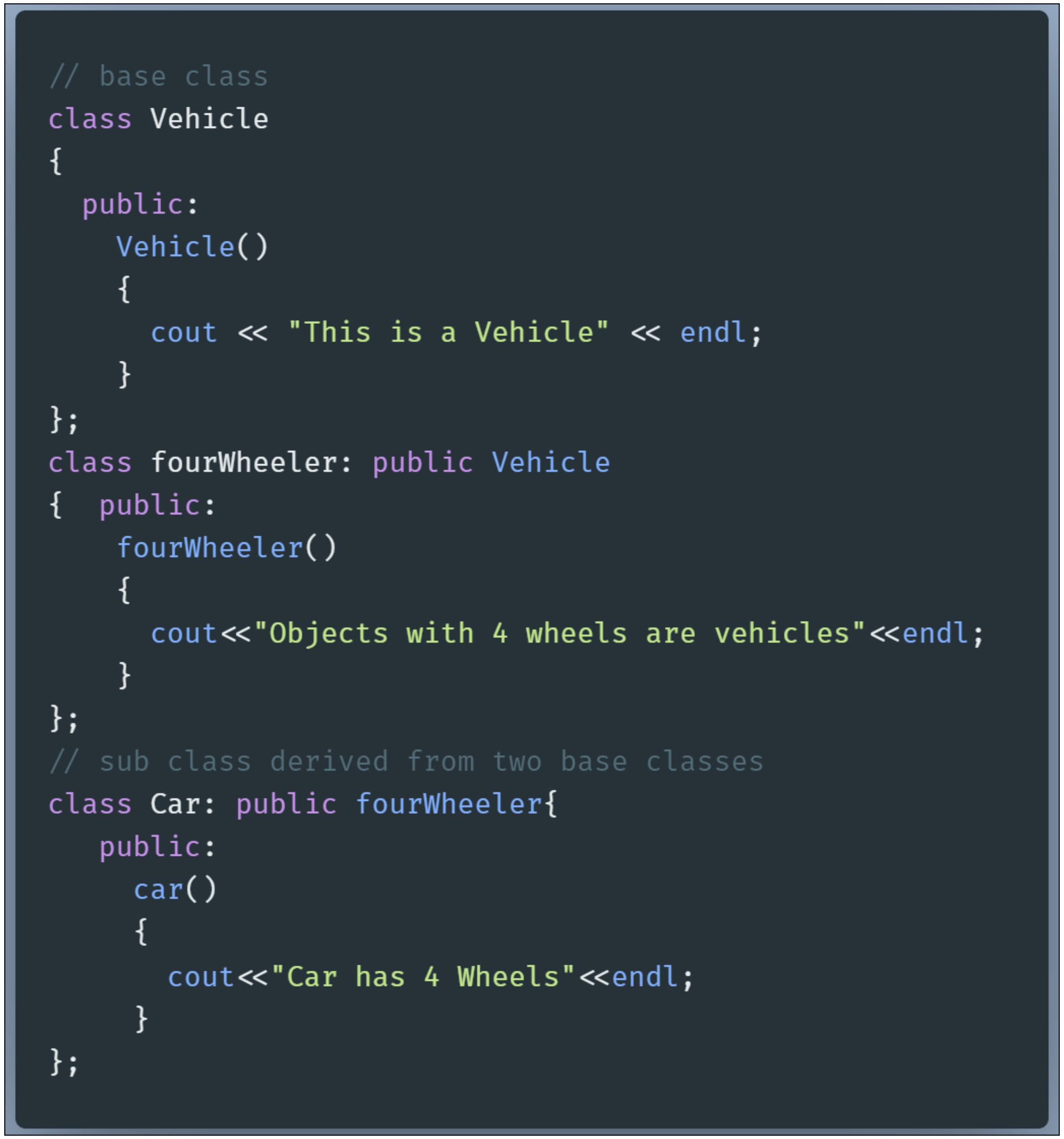
```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
  //body of subclass
};
```



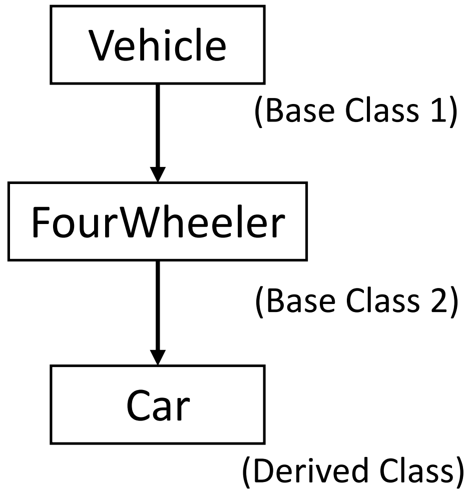
```
// first base class
class Vehicle
{
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};

// second base class
class FourWheeler
{
public:
    FourWheeler()
    {
        cout << "This is a 4 wheeler Vehicle" << endl;
    }
};

// sub class derived from two base classes
class Car : public Vehicle, public FourWheeler
{
};
```



Multi Level Inheritance



```
// base class
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};
class fourWheeler: public Vehicle
{  public:
    fourWheeler()
    {
      cout<<"Objects with 4 wheels are vehicles"<<endl;
    }
};
// sub class derived from two base classes
class Car: public fourWheeler{
   public:
     car()
     {
       cout<<"Car has 4 Wheels"<<endl;
     }
};
```



$$\begin{tikzcd}\boxed{\text{Vehicle}} \arrow[d, "\text{(Base Class 1)}"] \\ \boxed{\text{FourWheeler}} \arrow[d, "\text{(Base Class 2)}"] \\ \boxed{\text{Car}}\ \text{(Derived Class)}\end{tikzcd}$$