24.6 — Adding new functionality to a derived class

ALEX[1] SEPTEMBER 11, 2023

In the introduction to inheritance (https://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/)[2] lesson, we mentioned that one of the biggest benefits of using derived classes is the ability to reuse already written code. You can inherit the base class functionality and then add new functionality, modify existing functionality, or hide functionality you don't want. In this and the next few lessons, we'll take a closer look at how each of these things is done.

First, let's start with a simple base class:

```
#include <iostream>

class Base
{
protected:
    int m_value {};

public:
    Base(int value)
        : m_value { value }
    {
    }

    void identify() const { std::cout << "I am a Base\n"; }
};
```

Now, let's create a derived class that inherits from Base. Because we want the derived class to be able to set the value of m_value when derived objects are instantiated, we'll make the Derived constructor call the Base constructor in the initialization list.

```
class Derived: public Base
{
public:
    Derived(int value)
        : Base { value }
    {
    }
};
```

Adding new functionality to a derived class

In the above example, because we have access to the source code of the Base class, we can add functionality directly to Base if we desire.

There may be times when we have access to a base class but do not want to modify it. Consider the case where you have just purchased a library of code from a 3rd party vendor, but need some extra functionality. You could add to the original code, but this isn't the best solution. What if the vendor sends you an update? Either your additions will be overwritten, or you'll have to manually migrate them into the update, which is time-consuming and risky.

Alternatively, there may be times when it's not even possible to modify the base class. Consider the code in the standard library. We aren't able to modify the code that's part of the standard library. But we are able to inherit from those classes, and then add our own functionality into our derived classes. The same goes for 3rd party libraries where you are provided with headers but the code comes precompiled.

In either case, the best answer is to derive your own class, and add the functionality you want to the derived class.

One obvious omission from the Base class is a way for the public to access m_value. We could remedy this by adding an access function in the Base class -- but for the sake of example we're going to add it to the derived class instead. Because m_value has been declared as protected in the Base class, Derived has direct access to it.

To add new functionality to a derived class, simply declare that functionality in the derived class like normal:

```
class Derived: public Base
{
public:
    Derived(int value)
        : Base { value }
    {
    }

    int getValue() const { return m_value; }
};
```

Now the public will be able to call getValue() on an object of type Derived to access the value of m_value.

```
int main()
{
    Derived derived { 5 };
    std::cout << "derived has value " << derived.getValue() << '\n';

    return 0;
}
```

This produces the result:

```
derived has value 5
```

Although it may be obvious, objects of type Base have no access to the getValue() function in Derived. The following does not work:

```
int main()
{
    Base base { 5 };
    std::cout << "base has value " << base.getValue() << '\n';

    return 0;
}
```

This is because there is no getValue() function in Base. Function getValue() belongs to Derived. Because Derived is a Base, Derived has access to stuff in Base. However, Base does not have access to anything in Derived.

Next lesson
24.7 Calling inherited functions and overriding behavior

3

Back to table of contents

4

Previous lesson
24.5 Inheritance and access specifiers

5

6

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[8] are connected to your provided email address.

57 COMMENTS

Newest

Nidhi Gupta
May 6, 2025 9:05 am PDT

Base class and protected member:

cpp
Copy
Edit
class Base {
protected:
int m_value {};
public:

Base(int value) : m_value{ value } {}
void identify() const { std::cout << "I am a Base\n"; }
};
m_value is protected, so it can be accessed in derived classes, but not from outside the class (e.g., in main()).

identify() is a public method and can be accessed by both Base and Derived objects.

Derived class adds functionality:

cpp
Copy
Edit
class Derived : public Base {
public:
Derived(int value) : Base{ value } {}
int getValue() const { return m_value; }
};
Derived reuses Base's constructor to initialize m_value.

It introduces a new public method getValue() that allows access to the protected member.

0 Reply

EmtyC
December 24, 2024 11:37 am PST

I have a question:
Base class can't inherit from Derived (if Derived already inherits from Base), I tried it (by forward declaring Derived), and the only thing that happened was that the Base constructor complained that Derived doesn't have a default constructor. So, it's not possible by any mean is it?

Last edited 6 months ago by EmtyC

1 Reply

Alex Author
Reply to EmtyC [9] January 1, 2025 10:11 pm PST

No, because then you'd have a circular dependency.

1 Reply

learnccp lesson reviewer
July 24, 2023 1:38 pm PDT

However, Base does not have access to anything in Derived, its child as it can't take things from it's child. Base dont know of the existance of the child even, making it indpeendent from it. We can do anything in the derived as we can take things from our parent, but the parent dont take anything from the child, makes sense.

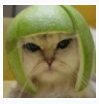
3 Reply

Emeka Daniel
April 27, 2023 3:18 pm PDT

Oh, yh I have been wanting to ask this question.

Regardless of which access specifier a member is under, when a derived inherits from a base, wouldn't the hidden const pointer be used for both members of the base and derived when an instantiation of an object of type derived is encountered?

1 Reply



Alex Author
Reply to Emeka Daniel [10] May 2, 2023 1:31 pm PDT

I'm not sure I understand what you mean. But let me try this:

The hidden `this` parameter always matches the type of the class the function is a member of.

If a derived function calls a base function, a pointer to the base portion of the derived object is passed as an argument. So basically, the base object has no idea whether it's being called on a standalone base object or a base object that is part of a derived.

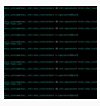
If you were asking something else, please clarify.

1 Reply

Emeka Daniel
Reply to Alex [11] September 5, 2023 3:04 pm PDT

Ok all clear now, thanks.
I must not received this email notification earlier, sorry.

0 Reply



learnccp lesson reviewer
Reply to Alex [11] July 24, 2023 1:38 pm PDT

this has a type?

1 Reply

Alex Author
Reply to learnccp lesson reviewer [12] July 28, 2023 5:26 pm PDT

Yes, all objects have a type. `this` has type `T* const`, where T is the class type.

1 Reply

EmtyC

Reply to Alex [13] December 24, 2024 11:25 am PST

oh man I still get confused on the fact that pointer can have the type of just declared types :)

0 Reply

Volker A4

April 21, 2022 3:22 pm PDT

This isn't really a question or anything, I just thought I'd share the example I made for myself this lesson.

```cpp
#include <iostream>

// We added print functionality through the derived class

class Base
{
    int m_value;

public:
    Base(int value)
        : m_value{ value }
    {
    }

    int get() const { return m_value; }
};

class Derived : public Base
{
public:
    Derived(int value)
        : Base{ value }
    {
    }

    friend std::ostream& operator<<(std::ostream& out, const Derived& d1);
};

std::ostream& operator<<(std::ostream& out, const Derived& d1)
{
    out << d1.get();

    return out;
}

int main()
{
    Derived derived{ 5 };

    std::cout << derived << '\n';

    return 0;
}
```

0 Reply

Sandersan

Reply to Volker A4 [14] August 13, 2022 12:51 am PDT

There is no need to make overloaded operator<< a friend member function, since the inherited function `get()` is also publicly accessible.

Last edited 2 years ago by Sandersan

3 Reply

Waldo Lemmer

July 30, 2021 1:08 pm PDT

- Variables should be initialized
- Classes and members should be initialized with {}

0 Reply

TimCook

November 23, 2020 10:13 am PST

>>Because Derived is a Base, Derived has access to stuff in Base. However, Base does not have access to anything in Derived.

shouldn't it be 'Because Derived is inherited from base...'?

0 Reply

nascardriver Sub-admin

Reply to TimCook [15] November 24, 2020 7:24 am PST

If `Derived` inherits from `Base`, `Derived` is a `Base`.

If `Pilot` inherits from `Human`, `Pilot` is a `Human`.

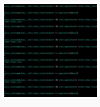
6 Reply

EmtyC

Reply to nascardriver [16] December 24, 2024 11:26 am PST

found you !!!!! :>

0 Reply



learnccp lesson reviewer

Reply to nascardriver [16] July 24, 2023 1:40 pm PDT

half human half pilot

1 Reply

Henrik

August 27, 2020 7:33 am PDT

I know in Swift you can make extensions of fundamental types like int or strings (or pretty much any class) and add your own functionality. Like so:

```
extension Int {
    func spellOutNumber() -> String {
        return NumberFormatter.localizedString(from: self as NSNumber, number:
.spellOut)
    }
}

print(1.spellOutNumber()) // prints "one"
```

Is it possible to do something similar in C++ by inheriting from, for example, std::string and implementing my own functionality? (I guess you can't inherit from fundamental types). Or is it against the "style" of C++, and I'm better off just making a whole new class that contains a std::string and methods to modify it?

Thanks for the tutorials btw, they are really great and easy to follow!

2 Reply

damon

Reply to Henrik [17] July 25, 2024 4:01 am PDT

Same with **extension methods** in C#

0 Reply

nascardriver Sub-admin

Reply to Henrik [17] August 27, 2020 7:53 am PDT

`std::string` isn't a fundamental type, you can inherit from it. Only the types that you don't need to include anything for are fundamental types. You can't inherit from those.

2 Reply

choofe

May 13, 2020 9:17 pm PDT

Hi
In the example on this lesson we make a way for public to access the protected member which I want to say is considered private for Base and the derived classes and also is of a base and I assume it should be one important class.
I wonder isn't it against the rule of thumb of encapsulation?
I mean we can write a function to modify the protected members,can't we?

```
class Derived: public Base
{
public:
    Derived(int value)
        :Base(value)
    {
    }

    int getValue() { return m_value; }
    void plusOne() { ++m_value; }
};
```

now we are able to modify the m_value in public.
The creator of the Base may want not m_value to be changed other than in derived class.
I think a clarify on this issue would be nice in the context as it may be mentioned in comments but you know... people like me may not go through all the comments.

0 Reply

Petter Nybråten

Reply to choofe [18] July 24, 2020 2:44 am PDT

Any m_value accessed through a Derived object belongs to that object, not a Base object.

0 Reply

Ayrton Fithiadi Sedjati

April 14, 2020 2:17 am PDT

I don't understand what you meant in the the last paragraph, "Because Derived is a Base, ...".

Did you mean "Because Derived is a *child of* Base, ..."?

Edit: Oh you might have meant "Because Derived is-a Base, ...". Am I correct?

0 Reply

nascardriver Sub-admin

Reply to Ayrton Fithiadi Sedjati [19] April 16, 2020 6:55 am PDT

Correct

0 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/
3. https://www.learncpp.com/cpp-tutorial/calling-inherited-functions-and-overriding-behavior/
4. https://www.learncpp.com/
5. https://www.learncpp.com/cpp-tutorial/inheritance-and-access-specifiers/
6. https://www.learncpp.com/adding-new-functionality-to-a-derived-class/
7. https://www.learncpp.com/cpp-tutorial/multiple-inheritance/
8. https://gravatar.com/
9. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-605675
10. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-579873
11. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-579979
12. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-584439
13. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-584684
14. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-567911
15. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-482386
16. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-482495
17. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-471654
18. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-463138
19. https://www.learncpp.com/cpp-tutorial/adding-new-functionality-to-a-derived-class/#comment-459948
20. https://g.ezoic.net/privacy/learncpp.com