8.4 — Access functions and encapsulation

BY ALEX ON SEPTEMBER 4TH, 2007 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

Why make member variables private?

In the previous lesson, we mentioned that class member variables are typically made private. Developers who are learning about object-oriented programming often have a hard time understanding why you'd want to do this. To answer that question, let's start with an analogy.

In modern life, we have access to many electronic devices. Your TV has a remote control that you can use to turn the TV on/off. You drive a car (or scooter) to work. You take a picture on your smartphone. All three of these things use a common pattern: They provide a simple interface for you to use (a button, a steering wheel, etc…) to perform an action. However, how these devices actually operate is hidden away from you. When you press the button on your remote control, you don't need to know what it's doing to communicate with your TV. When you press the gas pedal on your car, you don't need to know how the combustion engine makes the wheels turn. When you take a picture, you don't need to know how the sensors gather light into a pixellated image. This separation of interface and implementation is extremely useful because it allows us to use objects without understanding how they work. This vastly reduces the complexity of using these objects, and increases the number of objects we're capable of interacting with.

For similar reasons, the separation of implementation and interface is useful in programming.

Encapsulation

In object-oriented programming, **Encapsulation** (also called **information hiding**) is the process of keeping the details about how an object is implemented hidden away from users of the object. Instead, users of the object access the object through a public interface. In this way, users are able to use the object without having to understand how it is implemented.

In C++, we implement encapsulation via access specifiers. Typically, all member variables of the class are made private (hiding the implementation details), and most member functions are made public (exposing an interface for the user). Although requiring users of the class to use the public interface may seem more burdensome than providing public access to the member variables directly, doing so actually provides a large number of useful benefits that help encourage class re-usability and maintainability.

Note: The word encapsulation is also sometimes used to refer to the packaging of data and functions that work on that data together. We prefer to just call that object-oriented programming.

Benefit: encapsulated classes are easier to use and reduce the complexity of your programs

With a fully encapsulated class, you only need to know what member functions are publicly available to use the class, what arguments they take, and what values they return. It doesn't matter how the class was implemented internally. For example, a class holding a list of names could have been implemented using a dynamic array of C-style strings, std::array, std::vector, std::map, std::list, or one of many other data structures. In order to use the class, you don't need to know (or care) which. This dramatically reduces the complexity of your programs, and also reduces mistakes. More than any other reason, this is the key advantage of encapsulation.

All of the classes in the C++ standard library are encapsulated. Imagine how much more complicated C++ would be if you had to understand how std::string, std::vector, or std::cout were implemented in order to use them!

Benefit: encapsulated classes help protect your data and prevent misuse

Global variables are dangerous because you don't have strict control over who has access to the global variable, or how they use it. Classes with public members suffer from the same problem, just on a smaller scale.

For example, let's say we were writing a string class. We might start out like this:

```cpp
class MyString
{
    char *m_string; // we'll dynamically allocate our string here
    int m_length; // we need to keep track of the string length
};
```

These two variables have an intrinsic connection: m_length should always equal the length of the string held by m_string (this connection is called an **invariant**). If m_length were public, anybody could change the length of the string without changing m_string (or vice-versa). This would put the class into an inconsistent state, which could cause all sorts of bizarre problems. By making both m_length and m_string private, users are forced to use whatever public member functions are available to work with the class (and those member functions can ensure that m_length and m_string are always set appropriately).

We can also help protect the user from mistakes in using our class. Consider a class with a public array member variable:

```cpp
class IntArray
{
public:
    int m_array[10];
};
```

If users can access the array directly, they could subscript the array with an invalid index, producing unexpected results:

```cpp
int main()
{
    IntArray array;
    array.m_array[16] = 2; // invalid array index, now we overwrote memory that we don't own
}
```

However, if we make the array private, we can force the user to use a function that validates that the index is valid first:

```cpp
class IntArray
{
private:
    int m_array[10]; // user can not access this directly any more

public:
    void setValue(int index, int value)
    {
        // If the index is invalid, do nothing
        if (index < 0 || index >= 10)
            return;

        m_array[index] = value;
    }
};
```

In this way, we've protected the integrity of our program. As a side note, the at() functions of std::array and std::vector do something very similar!

Benefit: encapsulated classes are easier to change

Consider this simple example:

```cpp
#include <iostream>

class Something
{
public:
    int m_value1;
    int m_value2;
    int m_value3;
};

int main()
{
    Something something;
    something.m_value1 = 5;
    std::cout << something.m_value1 << '\n';
}
```

While this program works fine, what would happen if we decided to rename m_value1, or change its type? We'd break not only this program, but likely most of the programs that use class Something as well!

Encapsulation gives us the ability to change how classes are implemented without breaking all of the programs that use them.

Here is the encapsulated version of this class that uses functions to access m_value1:

```
#include <iostream>

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; }
};

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

Now, let's change the class's implementation:

```
#include <iostream>

class Something
{
private:
    int m_value[3]; // note: we changed the implementation of this class!

public:
    // We have to update any member functions to reflect the new implementation
    void setValue1(int value) { m_value[0] = value; }
    int getValue1() { return m_value[0]; }
};

int main()
{
    // But our program still works just fine!
    Something something;
    something.setValue1(5);
    std::cout << something.getValue1() << '\n';
}
```

Note that because we did not alter the prototypes of any functions in our class's public interface, our program that uses the class continues to work without any changes.

Similarly, if gnomes snuck into your house at night and replaced the internals of your TV remote with a different (but compatible) technology, you probably wouldn't even notice!

Benefit: encapsulated classes are easier to debug

And finally, encapsulation helps you debug the program when something goes wrong. Often when a program does not work correctly, it is because one of our member variables has an incorrect value. If everyone is able to access the variable directly, tracking down which piece of code modified the variable can be difficult (it could be any of them, and you'll need to breakpoint them all to figure out which). However, if everybody has to call the same public function to modify a value, then you can simply breakpoint that function and watch as each caller changes the value until you see where it goes wrong.

Access functions

Depending on the class, it can be appropriate (in the context of what the class does) for us to be able to directly get or set the value of a private member variable.

An **access function** is a short public function whose job is to retrieve or change the value of a private member variable. For example, in a String class, you might see something like this:

```
class MyString
{
private:
    char *m_string; // we'll dynamically allocate our string here
    int m_length; // we need to keep track of the string length

public:
    int getLength() { return m_length; } // access function to get value of m_length
};
```

getLength() is an access function that simply returns the value of m_length.

Access functions typically come in two flavors: getters and setters. **Getters** (also sometimes called **accessors**) are functions that return the value of a private member variable. **Setters** (also sometimes called **mutators**) are functions that set the value of a private member variable.

Here's a sample class that has getters and setters for all of its members:

```
class Date
{
private:
    int m_month;
    int m_day;
    int m_year;

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

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

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

The Date class above is essentially an encapsulated data struct with a trivial implementation, and a user of the class might reasonably expect to be able to get or set the day, month, or year.

The MyString class above isn't used just to transport data -- it has more complex functionality and has an invariant that needs to be maintained. No setter was provided for variable m_length because we don't want the user to be able to set the length directly (length should only be set whenever the string is changed). In this class, it does make sense to allow the user to get the string length directly, so a getter for the length was provided.

Getters should provide "read-only" access to data. Therefore, the best practice is that they should return by value or const reference (not by non-const reference). A getter that returns a non-const reference would allow the caller to modify the actual object being referenced, which violates the read-only nature of the getter (and violates encapsulation).

Best practice: Getters should return by value or const reference

Access functions concerns

There is a fair bit of discussion around in which cases access functions should be used or avoided. Although they don't violate encapsulation, some developers would argue that use of access functions violates good OOP class design (a topic that could easily fill an entire book).

For now, we'll recommend a pragmatic approach. As you create your classes, consider the following:

- If nobody outside your class needs to access a member, don't provide access functions for that member.
- If someone outside your class needs to access a member, think about whether you can expose an behavior or action instead (e.g. rather than a setAlive(bool) setter, implement a kill() function instead).
- If you can't, consider whether you can provide only a getter.

Summary

As you can see, encapsulation provides a lot of benefits for just a little bit of extra effort. The primary benefit is that encapsulation allows us to use a class without having to know how it was implemented. This makes it a lot easier to use classes we're not familiar with.

8.5 -- Constructors

Index

8.3 -- Public vs private access specifiers

C++ TUTORIAL | PRINT THIS POST

110 comments to 8.4 — Access functions and encapsulation

« Older Comments 1 2

Wallace
December 6, 2019 at 4:09 pm · Reply

Subject-verb agreement typo: "As a side note, the at() function of std::array and std::vector do something very similar!" should be "As a side note, the at() functions of std::array and std::vector do something very similar!"

Also, the introductory analogies may not be understood by younger readers. Replace "Your TV has a remote control that you can use to turn the TV on/off. You drive a car to work. You take a picture on your digital camera." with "Your smartphone has an app you open to watch videos. You open an app to hail a ride to work. You take a picture with an app on your phone." ;)

Alex
December 10, 2019 at 7:53 pm · Reply

Grammar fixed, and some of the analogy suggestions integrated. Thanks!

ADIL PHILIP MATHEW
October 22, 2019 at 11:14 pm · Reply

Getters should provide “read-only” access to data. Therefore, the best practice is that they should return by value or const reference (not by non-const reference). A getter that returns a non-const reference would allow the caller to modify the actual object being referenced, which violates the read-only nature of the getter (and violates encapsulation)
1.how is it violating the encapsulation, basically, in setters we are changing the value of private members how is it different from setters??

nascardriver
October 23, 2019 at 3:05 am · Reply

In a setter, you can at least verify that the passed in value is valid.

```
void setAge(int iAge)
{
  assert(iAge >= 0);
  // ...
}
```

With a non-const getter, you're giving the caller absolute control over the member. It's as if the member itself was `public`.

Dan
September 3, 2019 at 5:09 am · Reply

I have a really difficult time comprehending why anyone would say that access functions would be a violation of OOP, a violation of encapsulation, or bad in anyway for that matter. What's the point in any program if the user doesn't have access to view or modify the data in some way? Isn't part of the point to give access but limit it such that the program can't be broken and so that the class is portable to other programs? For example, a window object isn't very useful if I can't change its dimensions (i would assume one would want to make the dimensions private member variables with pre-defined max and min sizes). It seems to me that a program without access to data at all or only having variables and functions outside of the classes isn't as useful or portable as it could be. If you're going to say, "let's create a class but not give access to the data through functions", there is no point to using classes. Just use a struct. Maybe I'm overthinking or not able to think of situations where not having access to data would be useful?

Alex
September 4, 2019 at 10:59 am · Reply

https://dev.to/scottshipp/avoid-getters-and-setters-whenever-possible-c8m is a fairly comprehensible starter article on the topic, with interesting comments as well. The point isn't to needlessly limit access to a class, but rather to be more intentional about what kinds of public interfaces classes provide.

Dan
September 10, 2019 at 3:52 am · Reply

Thanks! Love all this. It’s a great help in learning this stuff!

Samira Ferdi
September 15, 2019 at 9:31 pm · Reply

So, you basically said, we have to define the needs of accessing. If we think that user need to access our data, we give them access' with appropriate way, otherwise we don't give any access at all! I think the idea of limited access of class (this is what I like about class) that user cannot change carelessly so user cannot ruin our class that can make our class broken or change the purpose of the class at the firstplace! Imagine if computer user gived all access of OS settings, than the computer must be crash!

ROHIT
August 11, 2019 at 3:58 am · Reply

what does it mean by "implementation of object" in the line "keeping the details about how an object is implemented hidden away from users of the object. Instead, users of the object access the object through a public interface." ????
is it means the member variables???

Alex
August 13, 2019 at 10:36 am · Reply

Yes, the "implementation" refers to the member variables that are typically kept non-public.

Nirbhay
August 8, 2019 at 1:30 am · Reply

Hello!

I don't understand how normal class member functions which are named 'public:'(Public interfaces) are any different from the "Access functions" (Getters and Setters).

What am I missing here?

Thank you.

nascardriver
August 8, 2019 at 6:00 am · Reply

Getters and settings get or set a member variable. Other member functions provide operations on the members.
`getDay` returns a member and does nothing else, it's a getter.
`setDay` sets a member and does nothing else, it's a setter.
`print` isn't a getter or setter.

Nirbhay
August 9, 2019 at 12:47 am · Reply

1. I see. But both 'print' and 'setDay' have access to the private member variables of the class, right? (I thought Access Functions = Member functions that can "access" the private member variables of the class.)
2. Does a class also have a private member function? and for which reason?

nascardriver
August 9, 2019 at 2:21 am · Reply

1.
There's no internal difference between the functions. "Access Function" means that it's used the access members.

2.
Yes.
In this example, you could have an `isValidDate` function. The the user of the class, every date is valid, so they don't need to have access to this function. It's only used internally, eg. when advancing the date.

Nirbhay
August 9, 2019 at 4:08 am · Reply

Alright thanks :)

Nirbhay
August 7, 2019 at 2:36 am · Reply

Hello!

Is there a difference between 'Access functions' and 'Public interfaces'?

Thannks :)

Alex
August 8, 2019 at 7:42 pm · Reply

The "Public interface" is any member function that has public access. "Access functions" are trivial functions that give you direct access to read or write members.

Access functions are typically part of the public interface.

Nirbhay
August 9, 2019 at 12:43 am · Reply

Thanks!

Piyush
June 17, 2019 at 6:49 am · Reply

I think.

"The Data class above is essentially an encapsulated data struct with a trivial implementation, and a user of the class might reasonably expect to be able to get or set the day, month, or year."

Should be :-

"The Date class above is essentially an encapsulated date struct with a trivial implementation, and a user of the class might reasonably expect to be able to get or set the day, month, or year."

Alex
June 19, 2019 at 1:43 pm · Reply

Fixed. Thanks!

Raga
May 12, 2019 at 7:19 am · Reply

I never really liked how getter setter looks lol, but I kinda understand when people say that exposing member variables is dangerous.

A good alternative my friend told me is the "tell, don't ask" principle, to give method rather than to let the user manipulate the state directly.

This link also taught me something about this : https://dev.to/scottshipp/avoid-getters-and-setters-whenever-possible-c8m

What do you think? Maybe this will be a good note about encapsulations for this page. Cheers.

Alex
May 16, 2019 at 4:41 pm · Reply

In principle, I agree with those recommendations. But in practice, I think this is one of those areas where it's better to be pragmatic than dogmatic, as I don't think those recommendations can be uniformly applied. For a complex classes, absolutely try to design them in a way that avoids getters and particularly setters in favor of functions that enable behaviors. But for simple classes like the Date class in this lesson, there isn't much benefit gained by avoiding access functions -- and access functions are preferable to Date being a plain old data struct that allows direct access to members.

Use the right tools and practices for the job at hand.

NXPY
March 30, 2019 at 4:49 am · Reply

Hi Alex !

Is std::string a class ? Are all functions accessed through member select operator part of a class ?

nascardriver
March 30, 2019 at 4:52 am · Reply

> Is std::string a class
Yes

> Are all functions accessed through member select operator part of a class
Or struct

NXPY
March 30, 2019 at 5:49 am · Reply

Thank you @nascardriver !

Matty J
February 5, 2019 at 5:33 pm · Reply

How'd you find out about the gnomes?

Alex
February 7, 2019 at 6:26 am · Reply

I just gnome these things.

Matty J
February 7, 2019 at 4:23 pm · Reply

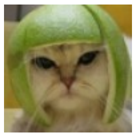
```
bool pun = false;
isPun("I just gnome these things.", pun); // function pass std::string and bool re
if (pun)
{ assert(pun && "I can't handle this."); }
elf
{ return 0;}
```

Nitin
September 11, 2018 at 12:35 pm · Reply

Alex,

As a way of saying thank you for your invaluable site, I'll list out the typos I stumbled upon in some of your chapters, starting with this chapter.

Typo - "In order to the use the class" --> In order to use the class.

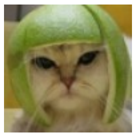


Alex
September 11, 2018 at 5:31 pm · Reply

Thanks for pointing out all of the typos across the various chapters!

Sam
September 8, 2018 at 5:42 pm · Reply

Typo: "We'd break not only this program, but likely most of programs that use class Something as well!" -> "We'd break not only this program, but likely most of THE programs that use class Something as well!"



Alex
September 10, 2018 at 12:52 pm · Reply

Fixed. Thanks!

Royd
September 8, 2018 at 12:01 pm · Reply

Hi Alex. Really excellent material. I notice you (quite suddenly I think) refer to "users (of the object)" in the context of encapsulation. I'm not quibbling about the nomenclature, but nodding to how important it becomes when there are "users" of the class, i.e. other programmers, not, well, "users" I guess. It's a huge subject but it might be a chapter in its own right to summarise how these and other mechanisms avoid multiple programmers stepping on each others toes! You have quite rightly referred to this type of issue many times, but perhaps there is a case for drawing those issues together? A new student, unused to many people working on the same code may be surprised at the importance of this. (I think Matteo was getting at the same point).

Nigel Booth
September 3, 2018 at 11:28 pm · Reply

Hi,

I know it's a little off the beaten track but wouldn't it be better to create a class using the class wizard in Visual Studio and then accessing it through the header file created for that class? Also, are virtual destructors necessary?

nascardriver
September 4, 2018 at 8:25 am · Reply

Hi Nigel!

> better to create a class using the class wizard in Visual Studio
Better, no. Faster, probably. But this tutorial isn't about IDE's, it's about the language. Most IDE's offer a way to generate code.

> are virtual destructors necessary?
Depends on your class. If the classes have memory that's not automatically managed, then yes, virtual destructors are necessary. Without them you'll get memory leaks.

pete
July 20, 2018 at 5:22 am · Reply

Hi Alex,
i've noticed the bodies of the access functions (mainly the setters and getters) are written with the same line of their declarations. is it something you would recommend to implement or just a means of demonstration?
thanks again.

Alex
July 24, 2018 at 3:00 pm · Reply

I typically put access functions definitions on the same line as the declarations to help keep the code vertically compact. This works because the bodies of the functions are so trivial (typically a single line).

I'm not aware of any reason to do otherwise, but your stylistic tastes may vary...

WiseFool
March 28, 2018 at 1:15 pm · Reply

I think it would sure be nice if, along with having a constructor and destructor with each class that - especially in std library classes - it would become a standard programming practice to include a function named help(), that when invoked, would simply print a list of public functions and members in that class, with parameters and a brief description of how they worked or what they were for. At least in utilty classes like cin, cout and string where it made sense to have a help() function.

nascardriver
March 29, 2018 at 1:31 am · Reply

Hi WiseFool!

That's what documentations are for
http://www.cplusplus.com/reference/istream/istream/
http://en.cppreference.com/w/cpp/io/basic_istream

WiseFool
March 31, 2018 at 11:32 am · Reply

Thank you, nascardriver, especially for the first reference. It gives excellent coverage of what's in there and how it works, with even sample programs. In just the few minutes that I looked at it this time I learned 2 or 3 new "tricks" that might be very handy to know some time. (I was aware of those 2 sites and had used them for quick explanations or syntax, but hadn't stumbled on the detailed coverage they give of std library classes and functions.)

Donlod
April 30, 2018 at 4:47 pm · Reply

Well i started learning programming with java and eclipse as IDE. And the java api is very well documented, what the methods are doing, what the params are for what return value to expect etc. and this is all build into the ide on mouseover. When I started with c++ this was no longer the case since the std lib functions mostly only provide type of params and return but no further information and sometimes those are even hard to read or some fancy typedefs. For example i still do not get all those makros and typedefs in the windows api, in the end the typedefs are just some kind of int and char. I think this is what I am missing the most when looking at java (as for now).

nascardriver
May 1, 2018 at 1:31 am · Reply

The standard library and Windows API are both documented, the documentations are just not integrated in the IDE. This would be a nice feature.

> fancy typedefs
You've probably encountered the STL (Standard template library).

> i still do not get all those makros and typedefs in the windows api
If you don't need Windows specific code don't use the Windows API. It's old code and inconsistent at times.

References
* http://www.cplusplus.com/reference/ (Better examples than cppreference, but slower)
* http://en.cppreference.com/w/cpp/header

Donlod
May 1, 2018 at 12:18 pm · Reply

> If you don't need Windows specific code don't use the Windows API. It's old code and inconsistent at times.
I was looking into xinput and windows to map xbox controller inputs to mouse and keyboard buttons.

Matteo
January 24, 2018 at 12:56 pm · Reply

I think that the best way to show the advantage of Encapsulation is by saying that it allows a clear separation between a user and a developer of a class. The developer is the only with the knowledge needed to work with the internals, and he/she offers an interface for the users to interact with such internals according to some use cases.

Luhan
October 11, 2017 at 1:41 pm · Reply

Can you implement a class from other file such as we do with the functions, for example, the forward declaration in the header file and the definition in a .cpp file?

Alex
October 12, 2017 at 4:07 pm · Reply

Yes! This is how classes are usually implemented, and I discuss how to do this later in this chapter.

AMG
August 15, 2017 at 12:26 pm · Reply

Hey Alex,
Typo: "Consider a class with an public array" -> "Consider a class with a public array". Your tutorial creates an illusion that programming is smooth, simple and easy.

Ivar
August 3, 2017 at 3:44 pm · Reply

In 8.2 — Classes and class members, you say that we should
"Use the struct keyword for data-only structures. Use the class keyword for objects that have both data and functions."

My question is, when would one have data-only structures? Doesn't this break the idea of encapsulation?

Alex
August 6, 2017 at 10:12 am · Reply

That's a surprisingly hard question to answer. Generally, you'll use structs when you need a light-weight (non-encapsulated) way to move data as a unit, and classes otherwise. It does break the idea of encapsulation, but sometimes that's okay, especially if we are defining something that isn't reusable, to solve the specific case of moving data from one place to another. In such a case, defining a fully encapsulated class can be a bit overkill.

Hamed O.Khaled
July 9, 2017 at 11:30 pm · Reply

Hey Alex!!
Can we say that the difference between the Abstraction and Encapsulation that:
In Abstraction we know little things that help us to use whatever is through an interface i.e. we know by pressing button on remote control specific functions will happen like changing channels or changing sound level.
In Encapsulation [[Access restricted]] we don't know anything about internals even we have no controls over them ((changing sound level through button as was in the Abstraction))
i.e. In T.V we don't know how is the display of the screen ((talking relative to the pixels)) is and we can't change in the mechanism on it.
am I Right ?
Thanks for this fabulous tutorials:)

Alex
July 10, 2017 at 8:21 am · Reply

There's a really good answer to this question **here**.

But yes, I think your analogy is correct.

« Older Comments 1 2