

#### In this Chapter:

- Unit 2. Encapsulation
- 2.3 Accessors & Mutators
  - What are accessors and mutators and why you should care
- Inline
- Constant methods
  - Warning!
  - Example 2.3 Time
  - Example 2.3 Card
  - Review 2
  - o Problem 3 6

## 2.3 Accessors & Mutators

Sue can see the value of designing class interfaces with the properties the client has in mind. However, she is very concerned about the resulting performance penalty. If a function call needs to be made every time the data from a class is accessed, won't this be a huge drain on the entire program? While mulling over this dilemma, Sam stops by and reminds Sue of the inline construct. "Yes!" she says, "that will definitely solve this problem..."

#### **Objectives**

By the end of this chapter, you will be able to:

- Make setters and getters inline to improve program performance
- Make getters constant to reduce the chance of accidentally modifying a member variable

#### **Prerequisites**

Before reading this chapter, please make sure you are able to:

- Recite and explain the four design rules for encapsulation (Chapter 2.0)
- Convert a UML class diagram into a class definition (Chapter 2.2)
- Implement member functions (Chapter 2.2)
- Use the this pointer to access member variables and methods (Chapte 2.2)
- Make a function inline using the inline keyword (Chapter 1.5)

#### What are accessors and mutators and why you should care

An accessor is a method providing the client access to data from member variables. Accessors are also commonly known as "getters" because the function names tend to be variations of get(). A Date class, for example, might have three accessors: getDay(), getMonth(), and getYear().

A mutator is a method enabling the client to modify or change the data stored in member variables. Mutators are also commonly known as "setters" because the function names tend to be variations of set(). A Date class, for example, might have four mutators: set() taking three parameters, setDay(), setMonth(), and setYear().

Just about every class you use or write will have some variation of a getter or setter. This should not be surprising: most classes store data and the client will want to have access to the data or change it. Because they are so important, it behooves us to learn how to make our getters and setters as safe and efficient as possible. That is the purpose of this chapter.

There are two tools we commonly use in getters and setters to accomplish these goals: making them inline and making them constant.

## **Inline**

As you may have noticed, most getters and setters tend to be trivial and consist of just a couple lines of code. This makes them great candidates for being inline methods. Recall that making a function inline is a signal to the compiler to make a special optimization. Instead of the function being called in the normal way, the compiler copies the function body code directly into that of the caller. While the behavior remains identical, performance is improved.  $\frac{2}{9}$ 

We can make a method inline though including the inline keyword just as we did with a standard function:

inline int Card : : getRank()	// the inline keyword makes this method
return value % 13;	// inline as it would for any other
	// function

Remember that inline functions need to go in the header file. This is true whether the inline function is a method or a stand-alone function.

It is so common to want to make a method inline (probably because so many methods are trivial) that C++ has made an easier way to do this. We can provide the body of a member function directly in the class definition, making the method inline.

class Card {	The inline keyword must be missing! If the method is defined in the class, it is inline automatically
public: void display();	// not inline because it is not trivial
// CARD : : GET RANK int getRank() return value % 13;	// use an abbreviated comment block. // traditionally, getters are even simpler // than setters because they don't // perform any validation
The "Card ::" is not used in the class definition	<pre>// use an abbreviated comment block. // because setRank() is a trivial function // consisting of just one statement,</pre>
value = rank * getSuit() * 13;	// it is a great candidate for inline
code removed for brevity	
private:	
bool validate();	// not inline because it is not trivial
int value;	// member variables are unchanged

## 4 Sue's Tips



It is not uncommon to have a class consisting completely of inline methods, making it unnecessary to write a source file: everything is in the header!

## **Constant methods**

As we discussed in Chapter 2.2, member variables have much larger scopes than local variables. Local variables are limited to the function in which they are defined but member variables are accessible to all the methods in the class. This is a potential source of bugs: the more accessible a variable is the larger the chance that it will be misused. If, for example, a local variable has an unexpected value that results in a bug,

the programmer need only look in the body of the function to find the source. However, if offending variable is a member variable, any of the methods in the class could be to blame.

Wouldn't it be great if we could restrict access to member variables, giving fewer methods access to make changes? That mechanism exists with the const method modifier.

When the const modifier is applied to a variable, the compiler provides guarantees that the variable will not be changed. When the const modifier is applied to a method, the compiler provides guarantees that the member variables will not be changed in the method.

If the programmer made a mistake and attempted to change value in the above function, the following compiler error would result:

```
In file included from cardTest.cpp:8:
card.h: In member function "int Card::getSuit() const":
card.h:50: error: assignment of data-member "Card::value" in read-only structur
```

# Warning!

The const modifier is a two-edged sword. It is a great help for finding bugs because the compiler will point to an unintended assignment (such as "if (value = 0)" where "if (value = 0)" was intended). However, it can also be really annoying. Consider the following function:

```
... code removed for brevity ...
bool Card :: validate(); // NOT const...
int getSuit() const // const so we cannot change value
{
    assert(validate()); // because validate() is not const, we cannot
    return value / 13; // guarantee that "value" isn't changed in
} // getSuit(). This will be a compile error
... code removed for brevity ...
```

We get a very cryptic error message:

```
In file included from cardTest.cpp:8:
card.h: In member function "int Card::getSuit() const":
card.h:50: error: passing "const Card" as "this" argument of "bool Card::validadiscards qualifiers
```

The problem here is that validate() is not a const method! Thus using const is an all-or-nothing proposition. A const method cannot call a method in the class that is not const.

#### Example 2.3 - Time

This example will demonstrate how to use the const and inline modifiers in a simple class definition. Most getters and setters should follow this pattern.

With the same UML diagram we used from Example 2.2, implement the getters and setters as inline and const as appropriate.

Time	
- minutes	
+ display	
+ set	
+ getMinutes	
+ getHours	
- validate	

All the code will be in the class definition so all the methods will be inline. Note that the functionality of the class will remain unchanged.

```
class Time
{
    public:
        // TIME :: SET : Set the time according to user input
        void set(int hours = 0, int minutes = 0)
        {
            // first just trust them
            this->minutes = hours * 60 + minutes;
            // set to midnight if invalid
            if (!validate())
                this->minutes = 0;
        }
    ... code removed for brevity...
        // TIME :: GET MINUTES : Fetch the minutes since midnight
        int getMinutes() const
        {
            // more paranoia...
            assert(validate());
            return minutes % 60;
        }
    ... code removed for brevity ...
        // TIME :: VALIDATE : Ensure that the member variable is set correctly
        bool validate() const // this must be const or it cannot be called by
        { // getMinutes() which is const!
            return (minutes >= 0 && minutes < 24 * 60);
        }
    };
```

As a challenge, modify the class so it also stores seconds as we did in Chapter 2.2. This means that the internal data representation will be "seconds since midnight" rather than "hours since midnight."

The complete solution is available at 2-3-time.html or: /home/cs165/examples/2-3-time.cpp

## Example 2.3 - Card

This example will demonstrate how to use the const and inline modifiers in a more complex class definition.

Write a class to represent a playing card. Include all the convenient getters and setters that the client may need to use this class.

The most interesting code for this problem is in the class definition in the card.h header.

```
#ifndef CARD_H
#define CARD_H
... code removed for brevity...
class Card
{
    public:
        // initialize
        void initialize() { value = 0; } // trivial functions can be on one lin
        // getters and setters
        bool setSuit(char chSuit);
        bool setRank(char chRank);
        // CARD :: GET SUIT fetch the suit {s,h,c,d}
        char getSuit() const
        {
            // paranoia
            assert(this->validate());
            return SUITS[value / 13];
        }
        // CARD :: GET RANK fetch the rank \{2,3,4,\ldots,j,q,k,a\}
        char getRank() const
            // paranoia
            assert(this->validate());
            return RANKS[value % 13];
        bool isInvalid() const { return value == INVALID; }
        void display();
    private:
        // holds the value. Though there are 256 possible, only 52 are used
        unsigned char value; //internal representation
    ... code removed for brevity...
#endif // CARD_H
```

The complete solution is available at 2-3-card.html or: /home/cs165/examples/2-3-card/

Given the following class:

```
class Coordinate
{
    public:
        int x;
    private:
        int y;
};
int main()
{
    Coordinate coord;
    coord.y = 3;
    cout << coord.y << endl;
    return 0;
}</pre>
```

#### **Review 2**

In the following class:

```
class Number
{
public:
    void set(int num)
    {
     <code goes here>
    }
private:
    int num;
};
```

What code will complete set ()?

### Problem 3 - 6

All four of these problems are to solve the same problem: create a class to hold an individual's grade on a test:  $(0 \to 100\% \text{ or } F \to A+)$ . Not that the properties must be both number and letter grades. It is up to the author of the class to determine the best data representation.

3. Create the UML class diagram to describe a class to hold an individual's grade

4. Write the class definition to match the above UML class diagram
5. Implement the method set() from your class definition
6. Write a driver program to exercise your Grade class