Learning Objectives

- **Define the term encapsulation**
- **Differentiate between `public` and `private`**
- **Explain which parts of a class should be public or private**
- **Explain why encapsulation can be beneficial**
- **Define data validation**

What is Encapsulation?

What is Encapsulation?

Encapsulation is a concept in which related data and methods are grouped together, and in which access to data is restricted. Grouping related data and functions makes thinking about your program a bit easier. Hiding or restricting how the user interacts with the data can keep the user from making unwanted changes.

The two main ideas of data restriction are the `public` and `private` keywords. These access modifiers (or keywords) can refer to classes, functions, and attributes. `public` means that the constructor, function, or attribute can be accessed both internally or externally. `private` means that the constructor, attribute, or function can only be accessed internally within the class itself.

Classes as Encapsulation

Classes in C++ are a form of encapsulation; they group together related data and functions. In the code below, the attributes `num1` and `num2` are grouped together with the functions `Describe` and `Sum`. They are all a part of `ExampleClass`. The instance `my_example` is not a part of the class itself; it is considered to be separate.

```
//add class definitions below this line

class ExampleClass {
    void SetN(int n1, int n2) {
      num1 = n1;
      num2 = n2;
    }

    void Describe() {
      cout << "My numbers are: " << num1 << " and " << num2 <<
        endl;
    }

    int Sum() {
      return num1 + num2;
    }

    int num1;
    int num2;
};

//add class definitions above this line
```

```
  //add code below this line

  ExampleClass my_example;
  my_example.SetN(5, 7);
  my_example.Describe();
  cout << my_example.Sum() << endl;

  //add code above this line
```

The code above results in an error, but we'll find out why next.

Hiding Data

You've learned in the Mutability chapter that class functions are typically `public` and attributes are typically `private`. The **public** access modifier enables other classes to have access to those functions. On the other hand, the **private** access modifier disables access to attributes to prevent users from imposing unwanted changes. Moving forward, we'll abide by the access modifier rules within the chart below.

Category	Public	Private
Constructor	X	

Functions	X	X
Attributes		X

▼ **Why are functions both public and private?**
In the pages that follow, you will see when making functions public is a good idea, and when keeping functions private is preferable. A well designed program will use a mix of public and private functions. Previously, we've learned that helper functions can be kept private since they are not directly accessed externally.

This is the same `ExampleClass` from above. It now uses the `public` and `private` access modifiers to hide the data or to make them accessible.

```
//add class definitions below this line

class ExampleClass {
  public:
    void SetN(int n1, int n2) {
      num1 = n1;
      num2 = n2;
    }

    void Describe() {
      cout << "My numbers are: " << num1 << " and " << num2 <<
        endl;
    }

    int Sum() {
      return num1 + num2;
    }

  private:
    int num1;
    int num2;
};

//add class definitions above this line
```

Note that when no access modifier is specified within a class, that attribute, function, or constructor is automatically declared as `private` by default. To make them public, you must specify `public` in the code.

Your code should run fine now because the instance or object is only interacting with `public` information. Now try to print the values for the `num1` and `num2` attributes.

```
//add code below this line

ExampleClass my_example;
my_example.SetN(5, 7);
cout << my_example.num1 << endl;
cout << my_example.num2 << endl;

//add code above this line
```

C++ produces an error message because an instance cannot directly access a `private` attribute. This is an example of hiding data. `my_example` cannot print `num1` or `num2` because they are private. However, `my_example` can access the public methods, which can then access the private attributes.

challenge

Try this variation:

- Create the functions `PrintNum1` and `PrintNum2` that print the `num1` and `num2` attributes.

▼ **Possible Solution**

```
void PrintNum1() {
  cout << num1 << endl;
}

void PrintNum2() {
  cout << num2 << endl;
}
```

```
ExampleClass my_example;
my_example.SetN(5, 7);
my_example.PrintNum1();
my_example.PrintNum2();
```

Public Access Modifier

Public Attributes

While C++ allows you to make instance attributes `public`, this is not encouraged. In fact, encapsulation asks that all attributes remain `private`. C++ itself, however, allows for public attributes. The following class has three attributes and all of them are public.

```
//add class definitions below this line

class Phone {
  public:
    Phone(string mo, int st, int me) {
      model = mo;
      storage = st;
      megapixels = me;
    }

    string model;
    int storage;
    int megapixels;
};

//add class definitions above this line
```

Instantiate a `Phone` object and manipulate the different attributes.

```
  //add code below this line

  Phone my_phone("iPhone", 256, 12);
  cout << my_phone.model << endl;
  my_phone.storage = 64;
  cout << my_phone.storage << endl;
  cout << my_phone.megapixels + 10 << endl;

  //add code above this line
```

When an attribute is public, a user can do whatever they want to it. This can become problematic. In the code above, the phone's storage was reduced by 75%. This should not happen. Encapsulation limits what and

how information is modified. By hiding data, you can ensure that users only manipulate the class in an approved manner.

Public Functions

Since all attributes should be private, we will use this access modifier for the following code sample. Unlike attributes, you are encouraged to have public functions. If all of your functions are private, it would be impossible to interact with the object. The constructor is a special kind of function, and this too should be public.

```
//add class definitions below this line

class Phone {
  public:
    Phone(string mo, int st, int me) {
      model = mo;
      storage = st;
      megapixels = me;
    }

  private:
    string model;
    int storage;
    int megapixels;
};

//add class definitions above this line
```

The real benefit of a public function is that it can access private attributes. Public functions are the gateway to dealing with private data. Create the public function `Describe` that prints a description of the `Phone` object.

```
//add class definitions below this line

class Phone {
  public:
    Phone(string mo, int st, int me) {
      model = mo;
      storage = st;
      megapixels = me;
    }

    void Describe() {
      cout << "My " << storage << " gig " << model;
      cout << " has a " << megapixels << " megapixels camera."
        << endl;
    }

  private:
    string model;
    int storage;
    int megapixels;
};

//add class definitions above this line
```

Instantiate a `Phone` object and call the `describe` method.

```
  //add code below this line

  Phone my_phone("iPhone", 256, 12);
  my_phone.Describe();

  //add code above this line
```

challenge

Try this variation

- Change the access modifier from `public` to `private`.

▼ **Why does this not work?**

The constructor is a special kind of function that is called when an object is created. Once the constructor is private, it cannot be called outside the class. That is why C++ throws an error message. The only way a private constructor can work is if a class is declared inside another class. The outer class can call the inner constructor even if it is private.

Private Access Modifier

As discussed on the previous page, we will be making all attributes private. Instance attributes, static attributes, constants — it does not matter, they will all be private.

Private Functions

Functions too, can be private. And just like private attributes, private functions are accessed by public functions. Here is an example class with a private function.

```
//add class definitions below this line

class PrivateExample {
  public:
    PrivateExample(int n) {
      num = n;
    }

    void PublicFunction() {
      PrivateFunction();
    }

  private:
    int num;

    void PrivateFunction() {
      cout << "The double of " << num << " is: " << num * 2 <<
        endl;
      cout << num << " squared is: " << num * num << endl;
    }
};

//add class definitions above this line
```

Instantiate an object and try to call `PrivateFunction`.

```
//add code below this line

PrivateExample my_example(5);
my_example.PrivateFunction();

//add code above this line
```

C++ throws an error message because an instance cannot directly access a private function. Change the function call to `PublicMethod` and run the code again. This time it should work because public functions can access private functions and/or attributes.

```
//add code below this line

PrivateExample my_example(5);
my_example.PublicFunction();

//add code above this line
```

Public and Private Methods

A well written C++ program will make use of both public and private functions. Deciding what to make public and what to make private comes down to how you want the user to interact with your code. Only make public those functions you want the user to call. Keep everything else private. The example below is a class that counts the number of vowels in a strings.

First let's add the private functions and attributes. Note that the `IsVowel` and `CountVowels` functions will serve as helper functions in a later function. This is why they do not need to be accessed directly by the user and should be `private`.

```
//add class definitions below this line

class Words {
  private:
    string word;
    int count;

    bool IsVowel(char ch) {
      ch = toupper(ch);
      return (ch=='A' || ch=='E' || ch=='I' ||
              ch=='O' || ch=='U');
    }

    int CountVowels(string str, int n) {
      if (n == 1) {
        return IsVowel(str[n-1]);
      }
      return CountVowels(str, n-1) + IsVowel(str[n-1]);
    }
};

//add class definitions above this line
```

Next, add the `public` components such as the constructor `Words`. The constructor instantiates an object with the `word` and `count` attributes. The other `public` component is the `Printer` function, which calls the `IsVowel` and `CountVowels` functions. Without this public `Printer` function, `IsVowel` and `CountVowels` will be completely inaccessible.

```
//add class definitions below this line

class Words {
  public:
    Words(string str, int n) {
      word = str;
      count = n;
    }

    void Printer() {
      cout << "The number of vowels in " << word;
      cout << " is " << CountVowels(word, count) << endl;
    }

  private:
    string word;
    int count;

    bool IsVowel(char ch) {
      ch = toupper(ch);
      return (ch=='A' || ch=='E' || ch=='I' ||
              ch=='O' || ch=='U');
    }

    int CountVowels(string str, int n) {
      if (n == 1) {
        return IsVowel(str[n-1]);
      }
      return CountVowels(str, n-1) + IsVowel(str[n-1]);
    }
};

//add class definitions above this line
```

Finally, instantiate a `string` to use as an argument and a `Words` object in order to run the function `Printer` and see the program output.

```
  //add code below this line

  string s = "house";
  Words vowels(s, s.length());
  vowels.Printer();

  //add code above this line
```