# Learning Objectives

- **Define the term encapsulation**
- **Differentiate between `public` and `private`**
- **Explain which parts of a class should public, private, or both**
- **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 methods 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 `public` and `private`. These access modifiers (or keywords) can refer to classes, methods, and attributes. Public means that the constructors, methods, or attributes can be accessed by an instance of a class. Private means that the constructor, attribute, or method can only be accessed by the class itself.

## Classes as Encapsulation

Classes in Java are a form of encapsulation; they group together related data and methods. In the image below, the attributes `num1` and `num2` are grouped together with the methods `describe` and `sum`. They are all a part of `ExampleClass` (highlighted in red). The instance `myExample` is not a part of the class itself; it is considered to be separate.

```
class ExampleClass {
    int num1;
    int num2;

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

    String describe() {
        return ("My numbers are: " + num1 + " and " + num2);
    }

    int sum() {
        return num1 + num2;
    }
}
```

```
ExampleClass myExample = new ExampleClass(5, 7);
System.out.println(myExample.describe());
System.out.println(myExample.sum());
```

Classes as Encapsulation

## Hiding Data

Up until now, we have been creating classes without using `public` or `private` keywords. When no access modifiers are used, Java assumes the `default` keyword. This means we have not fully implemented encapsulation because we did not put any restrictions on accessing the class. From now on, **all** classes will use `public` and `private` access modifiers. The chart below gives a brief overview of when to use each keyword.

| Category | Public | Private |
|---|---|---|
| Constructor | X | |
| Methods | X | X |
| Attributes | | X |

▼ **Why are methods both public and private?**
In the pages that follow, you will see when making methods public is a good idea, and when keeping methods private is preferable. A well designed program will use a mix of public and private methods.

This is the `ExampleClass` from the image above. It now uses the `public` and `private` access modifiers to hide data.

```
//add class definitions below this line

class ExampleClass {
  private int num1;
  private int num2;

  public ExampleClass(int n1, int n2) {
    num1 = n1;
    num2 = n2;
  }

  public String describe() {
    return ("My numbers are: " + num1 + " and " + num2);
  }

  public int sum() {
    return num1 + num2;
  }
}

//add class definitions above this line
```

Now instantiate an object of the `ExampleClass` and call the `describe` and `sum` methods. Run the program.

```
    //add code below this line

    ExampleClass myExample = new ExampleClass(5, 7);
    System.out.println(myExample.describe());
    System.out.println(myExample.sum());

    //add code above this line
```

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

```
    //add code below this line

    ExampleClass myExample = new ExampleClass(5, 7);
    System.out.println(myExample.num1);
    System.out.println(myExample.num2);

    //add code above this line
```

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

challenge

## Try this variation:

- Create the methods `printNum1` and `printNum2` that print the `num1` and `num2` attributes.

▼ **Possible Solution**

```
    public void printNum1() {
     System.out.println(num1);
    }

    public void printNum2() {
      System.out.println(num2);
    }
```

# Public Access Modifier

## Public Attributes

While Java allows you to make instance attributes public, this is not encouraged. In fact, encapsulation asks that all attributes remain private. Java 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 String model;
  public int storage;
  public int megapixels;

  Phone(String mo, int st, int me) {
    model = mo;
    storage = st;
    megapixels = me;
  }
}

//add class definitions above this line
```

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

```
    //add code below this line

    Phone myPhone = new Phone("iPhone", 256, 12);
    System.out.println(myPhone.model);
    myPhone.storage = 64;
    System.out.println(myPhone.storage);
    System.out.println(myPhone.megapixels + 10);

    //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.

▼ **Do `public` and `default` mean the same thing?**
Prior to learning about encapsulation, we used the `default` access modifier. We could modify the object any way we wanted. Similarly, we can modify the object any way you want with the `public` access modifier. Does this mean that `default` and `public` are the same? No. However, you will not notice a difference until you start working with packages, something we have not yet done. Elements with the `default` access modifier are only accessible in the same package.

## Public Methods

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 methods. If all of your methods are private, it would be impossible to interact with the object. The constructor is a special kind of method, and this too should be public.

```
//add class definitions below this line

class Phone {
  private String model;
  private int storage;
  private int megapixels;

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

//add class definitions above this line
```

The real benefit of a public method is that they can access private attributes. Public methods are the gateway to dealing with private data. Create the public method `describe` that prints a description of the `Phone` object.

```
//add class definitions below this line

class Phone {
  private String model;
  private int storage;
  private int megapixels;

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

  public void describe() {
    System.out.println("My " + storage + " gig " + model + " has
        a " + megapixels + " megapixel camera.");
  }
}

//add class definitions above this line
```

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

```
    //add code below this line

    Phone myPhone = new Phone("iPhone", 256, 12);
    myPhone.describe();

    //add code above this line
```

challenge

## Try this variation

- Change the constructor access modifier to `private`.

▼ **Why does this not work?**

The constructor is a special kind of method that is automatically called when the `new` keyword is used. Once the constructor is private, it cannot be called outside the class. That is why Java 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 Methods

Methods too, can be private. And just like private attributes, private methods are accessed by public methods. Here is an example class with a private method.

```
//add class definitions below this line

class PrivateExample {
  private int num;

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

  public void publicMethod() {
    privateMethod();
  }

  private void privateMethod() {
    System.out.println("The double of " + num + " is: " + num *
        2);
    System.out.println(num + " squared is: " + num * num);
  }
}

//add class definitions above this line
```

Instantiate an object and try to call `privateMethod`.

```
    //add code below this line

    PrivateExample myExample = new PrivateExample(5);
    myExample.privateMethod();

    //add code above this line
```

Java throws an error message because an instance cannot directly access a private method. Change the method call to `publicMethod` and run the code again. This time it should work because public methods can access private methods and/or attributes.

```
    //add code below this line

    PrivateExample myExample = new PrivateExample(5);
    myExample.publicMethod();

    //add code above this line
```

## Public and Private Methods

A well written Java program will make use of both public and private methods. 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 methods you want the user to call. Keep everything else private. The example below is a class that counts the number of vowels in a list of strings.

```
//add class definitions below this line

class Words {
  private String[] words;
  private int[] count;

  public Words(String[] listOfWords) {
    words = listOfWords;
    count = new int[words.length];
  }

  public void countVowels() {
    char[] vowels = {'a', 'e', 'i', 'o', 'u'};
    for (int i = 0; i < words.length; i++) {
      String word = words[i];
      for (int j = 0; j < word.length(); j++) {
        for (int k = 0; k < vowels.length; k++) {
          if (word.charAt(j) == vowels[k]) {
            count[i]++;
          }
        }
      }
    }
  }

  public void printVowelCount() {
    for (int num : count) {
      System.out.print(num + " ");
    }
    System.out.println();
  }
}

//add class definitions above this line
```

Instantiate a `Words` object with a list of strings. Then call the `countVowels` and `printVowelCount` methods. Your results should be `3 1 3 2`.

```
    //add code below this line

    String[] wordList = {"house", "cat", "computer", "Java"};
    Words w = new Words(wordList);
    w.countVowels();
    w.printVowelCount();

    //add code above this line
```

Both methods are public because the user is expected to use both of them. However, the `countVowels` method is hard to follow. For the sake of readability, we can create the helper method `isVowel`. The user is not expected to use this method, so make it private. The results should be the same.

```
public void countVowels() {
  for (int i = 0; i < words.length; i++) {
    String word = words[i];
    for (int j = 0; j < word.length(); j++) {
      if (isVowel(word.charAt(j))) {
        count[i]++;
      }
    }
  }
}

private boolean isVowel(char letter) {
  char[] vowels = {'a', 'e', 'i', 'o', 'u'};
  for (int i = 0; i < vowels.length; i++) {
    if (letter == vowels[i]) {
      return true;
    }
  }
  return false;
}
```

challenge

## Try these variations:

- Create the helper method `checkWord` that iterates through a string and determines if each character is a vowel or not. Your `countVowels` method should now look like this.

```
public void countVowels() {
  for (int i = 0; i < words.length; i++) {
    String word = words[i];
    checkWord(i, word);
  }
}
```

▼ **Possible Solution**

Here is one way to create the `checkWord` method. Remember to make this method private as it not to be called by the user.

```
  private void checkWord(int index, String word) {
    for (int i = 0; i < word.length(); i++) {
      char letter = word.charAt(i);
      if (isVowel(letter)) {
        count[index]++;
      }
    }
  }
```

# Code as a Black Box

## What is a Black Box?

The term “black box” is used to describe a system in which the internal workings are hidden from the user. In fact, the user is not expected to know how the system works; they only need to know how to use it. Encapsulation allows you to create classes that are black boxes. For example the `Words` class from the previous page is an example of this. By the time you finished the page, you transformed the code on the left to the code on the right.

**Before**

```
public void countVowels() {
  char[] vowels = {'a', 'e', 'i', 'o', 'u'};
  for (int i = 0; i < words.length; i++) {
    String word = words[i];
    for (int j = 0; j < word.length(); j++) {
      for (int k = 0; k < vowels.length; k++) {
        if (word.charAt(j) == vowels[k]) {
          count[i]++;
        }
      }
    }
  }
}
```

**After**

```
public void countVowels() {
  for (int i = 0; i < words.length; i++) {
    String word = words[i];
    checkWord(i, word);
  }
}

private void checkWord(int index, String word) {
  for (int i = 0; i < word.length(); i++) {
    char letter = word.charAt(i);
    if (isVowel(letter)) {
      count[index]++;
    }
  }
}

private boolean isVowel(char letter) {
  char[] vowels = {'a', 'e', 'i', 'o', 'u'};
  for (int i = 0; i < vowels.length; i++) {
    if (letter == vowels[i]) {
      return true;
    }
  }
  return false;
}
```

.guides/img/encapsulation/small-black-box

From the user’s point of view, there is no difference in how they interact with the class. In both cases they call the `countVowels` method. The user does not need to know about any of the changes because they cannot access private methods. The `Words` class is a black box. You should strive to use encapsulation to create classes that are black boxes.

## Creating a Black Box

We are going to create the black box class `Numbers`. This class requires a tab-delimited CSV file with numbers in it. These numbers are integers between 1 and 100 (including both 1 and 100). The only public method is `summary` which will print a summary of the numbers in the CSV file. Start by creating the class and the constructor.

```
//add class definitions below this line

class Numbers {
  private String file;
  private int[] data;

  public Numbers(String f) {
    file = f;
    data = getData();
  }
}

//add class definitions above this line
```

Notice how the `data` attribute calls the `getData` method. This is because the file needs to be automatically read and converted into an array of integers when a `Numbers` object is instantiated. The `getData` method should be private because it is a helper method. Most of this method comes from the previous lesson on reading files. Remember, the `readLine` method returns a single string of all the numbers read from the CSV file. The `split` method turns that single string into an array of strings. However, we need an array of integers, so the `convertToInts` method is called to make this transformation.

```
  private int[] getData() {
    int[] intTokens = new int[0];
    try {
      BufferedReader reader = new BufferedReader(new
        FileReader(file));
      String line = reader.readLine();
      String[] stringTokens = line.split("\t");
      intTokens = convertToInts(stringTokens);
      reader.close();
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      System.out.println("Finished reading a file.");
      return intTokens;
    }
  }
```

The `convertToInts` method creates an array of ints that is the same length as the array of strings. It then iterates over the array of strings, taking each element and converting it to an integer. This integer is then stored in the `ints` array.

```
private int[] convertToInts(String[] strings) {
  int size = strings.length;
  int[] ints = new int[size];
  for (int i = 0; i < size; i++) {
    ints[i] = Integer.parseInt(strings[i]);
  }
  return ints;
}
```

▼ **What are all of these methods doing?**

To create the value for the `data` attribute, the constructor calls the `getData` method. This method takes an array of strings and passes it to the `convertToInts` method. This method converts the array of strings into an array of integers. The integer array is returned to the `getData` method, which returns the same integer array to the `data` attribute.

We are now ready to make the public method `summary`. This method is going to provide a summary of information about the numbers in the CSV file. Start by calculating how many numbers there are.

```
public void summary() {
  printCount();
}

private void printCount() {
  System.out.println("There are " + data.length + "
      numbers.");
}
```

Instantiate a `Numbers` object and pass it a file path to a CSV file. Calling the `summary` method should prin that there are 37 numbers in the file.

```
  //add code below this line

  String path = "studentFolder/numbers.csv";
  Numbers myNumbers = new Numbers(path);
  myNumbers.summary();

  //add code above this line
```

▼ **Code**

```
import java.io.*;
```

```
//add class definitions below this line

class Numbers {
  private String file;
  private int[] data;

  public Numbers(String f) {
    file = f;
    data = getData();
  }

  private int[] getData() {
    int[] intTokens = new int[0];
    try {
      BufferedReader reader = new BufferedReader(new
      FileReader(file));
      String line = reader.readLine();
      String[] stringTokens = line.split("\t");
      intTokens = convertToInts(stringTokens);
      reader.close();
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      System.out.println("Finished reading a file.");
      return intTokens;
    }
  }

  private int[] convertToInts(String[] strings) {
    int size = strings.length;
    int[] ints = new int[size];
    for (int i = 0; i < size; i++) {
      ints[i] = Integer.parseInt(strings[i]);
    }
    return ints;
  }

  public void summary() {
    printCount();
  }

  private void printCount() {
    System.out.println("There are " + data.length + "
      numbers.");
  }
}

//add class definitions above this line
```

```
public class BlackBox {
  public static void main(String[] args) {

    //add code below this line

    String path = "studentFolder/numbers.csv";
    Numbers myNumbers = new Numbers(path);
    myNumbers.summary();

    //add code above this line
  }
}
```

challenge

## Try these variations:

- Create the `printNumbers` method that prints all of the numbers in the file.

▼ **Possible Solution**

Here is one possible solution for the `printNumbers` method:

```
  public void summary() {
    printCount();
    printNumbers();
  }

  private void printNumbers() {
    System.out.print("The numbers are: ");
    for (int num : data) {
      System.out.print(num + " ");
    }
    System.out.println();
  }
```

- Create the `printSmallest` method that prints the smallest number in the file.

▼ **Possible Solution**

Here is one possible solution for the `printSmallest` method:

```
public void summary() {
  printCount();
  printNumbers();
  printSmallest();
}

private void printSmallest() {
  int smallest = 100;
  for (int num : data) {
    if (num < smallest) {
      smallest = num;
    }
  }
  System.out.println("The smallest number is " +
    smallest + ".");
}
```

- Create the `printLargest` method that prints the largest number in the file.

▼ **Possible Solution**

Here is one possible solution for the `printLargest` method:

```
public void summary() {
  printCount();
  printNumbers();
  printSmallest();
  printLargest();
}

private void printLargest() {
  int largest = 1;
  for (int num : data) {
    if (num > largest) {
      largest = num;
    }
  }
  System.out.println("The largest number is " + largest
    + ".");
}
```