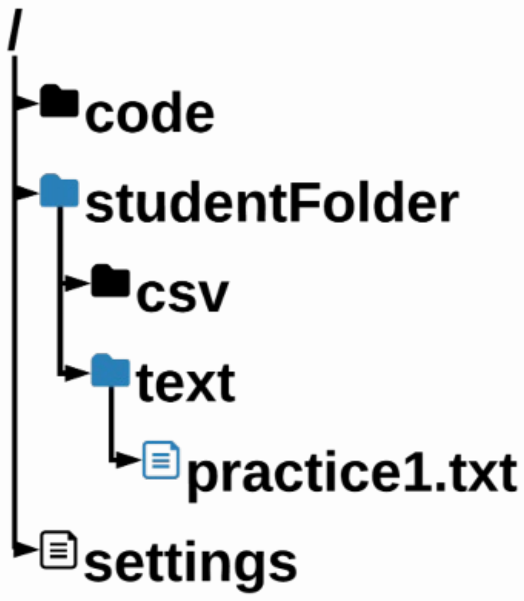
Learning Objectives - Writing

- Navigate the file system to the appropriate folder
- Explain why using a buffer is preferable to interacting directly with the disk
- Explain what happens when you write a file that does not exist
- Demonstrate how to write multiline strings to a file
- Differentiate between write and append modes

File Basics

File Basics

This module is all about working with files on a computer. The first step is to locate the desired file. That means being able to navigate the file system. The file we are going to use is called `practice1.txt`. It is located in the `text` folder, which is inside the folder called `studentFolder`. So the path to the file is: `studentFolder/text/practice1.txt`.



File Path

Use a string to represent the file path. This string will be passed to the objects used to read and write to files.

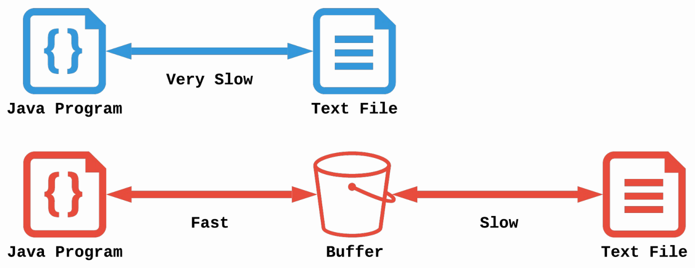
```
//add code below this line
String path = "studentFolder/text/practice1.txt";

//add code above this line
```

Buffers

The most basic ways to read and write to files are with the `FileReader` and `FileWriter` classes. However, they only read and write one byte at a time from disk. A byte is not a large amount of data. So programs with `FileReader` and `FileWriter` have to access the disk many times. This is not an efficient way to work with files.

Using a buffer can speed up your program. A buffer is a block of memory that stores data. Your program interacts with the buffer (which is fast) before interacting with the disk (which is slow). Buffers can read or write more than one byte of data at a time, so it interacts with the disk less often.



Buffer

The examples in this module will always use a buffer. We will start with writing to a file. Use the `FileWriter` class, but "wrap" it in the `BufferedWriter` class. This allows you to send data to a buffer (fast) before writing to the file (slow). Be sure to import the `java.io` package.

```
//add code below this line
String path = "studentFolder/text/practice1.txt";
BufferedWriter writer = new BufferedWriter(new
    FileWriter(path));


//add code above this line
```

IO Exceptions

Running the code above generates an error message. Your programs that read and write to files need to be able to handle input/output exceptions. An input/output exception can occur for a variety of reasons. Here are some cases in which Java would throw an exception:

- You are reading/writing to a file on the network, and the connection was lost
- You are reading/writing to a local file that is no longer available
- You are reading/writing to a file but do not have permission
- You are writing to a file but do not have enough disk space

Use a `try... catch` block to handle input/output exceptions. In the `try` portion of the block, create a `BufferedWriter` object. If there is an exception, print it in the `catch` portion of the block. This code should run without any errors, but it does not do anything.

```java
//add code below this line
String path = "studentFolder/text/practice1.txt";
try {
  BufferedWriter writer = new BufferedWriter(new
    FileWriter(path));
} catch (IOException e) {
  System.out.println(e);
}
//add code above this line
```

Writing to a File

Writing to a File

Continuing from the previous page, your program should import the `java.io` package, have the file path stored as a string, use a `BufferWriter` object, and catch any input/output exceptions.

```
//add code below this line
String path = "studentFolder/text/practice1.txt";
try {
  BufferedWriter writer = new BufferedWriter(new
    FileWriter(path));
} catch (IOException e) {
  System.out.println(e);
}
//add code above this line
```

If there is no input/output exception, create a string with the text you want to write to the file. Then use the `write` method to write this text to the file. Always close the file once you are done writing to it. Add a `finally` block so the user gets a message that the program has finished writing to file. After running the program, you can click the link to open the file and see the message.

```
//add code below this line
String path = "studentFolder/text/practice1.txt";
try {
  BufferedWriter writer = new BufferedWriter(new
    FileWriter(path));
  String text = "Hello there";
  writer.write(text);
  writer.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished writing to a file.");
}
//add code above this line
```

Open practice1.txt

challenge

Try these variations:

Be sure to open the `practice1.txt` file after each change.
* Change the string `text` to `"Goodbye"`
* Change the string `text` to `""`
* Open `studentFolder` in the sidebar on the left. Open the `text` folder and right-click on `practice1.txt`. Select “Delete…” and run the program again.

▼ **Why is there no error message?**
If you tell Java to write to a nonexistent file (the third suggestion), it will create the file for you. That is why you do not see an error message. `FileWriter` will throw an input/output exception if the path is a directory rather than a regular file, if the file does not exist but cannot be created, or if the file cannot be opened for any other reason.

Open practice1.txt

Multiline Strings

Multiline Strings

Let’s take the Java code from the previous page and make a few changes. This program will write to the file `practice2.txt`, and there will be two strings to write. The first string, `text1`, has the same value as before. The string `text2` has the value `my friend`. Write both of these strings to the file; then click on the link.

```
    //add code below this line
    String path = "studentFolder/text/practice2.txt";
    try {
      BufferedWriter writer = new BufferedWriter(new
        FileWriter(path));
      String text1 = "Hello there";
      String text2 = "my friend";
      writer.write(text1);
      writer.write(text2);
      writer.close();
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      System.out.println("Finished writing to a file.");
    }
    //add code above this line
```

Open practice2.txt

Notice how Java writes text as one string after another; it does not add a new line, it does not add a space. You need to add these characters yourself. Change the string variables to look like the code below. Run the program, and then look at the file. Each string should be on its own line.

```
      String text1 = "Hello there\n";
      String text2 = "my friend";
```

Open practice2.txt

Another benefit of the `BufferedWriter` class is that it has the method `newLine` which writes a line separator. You can use this method instead of using the newline character in your strings.

```
//add code below this line
    String path = "studentFolder/text/practice2.txt";
    try {
      BufferedWriter writer = new BufferedWriter(new
        FileWriter(path));
      String text1 = "Hello there";
      String text2 = "my friend";
      writer.write(text1);
      writer.newLine();
      writer.write(text2);
      writer.close();
    } catch (IOException e) {
      System.out.println(e);
    } finally {
      System.out.println("Finished writing to a file.");
    }
    //add code above this line
```

Open practice2.txt

Substrings

You can also write a part of a string (called a substring) to a file. The `write` method can also take two additional parameters. The first is an integer representing the starting character, and the second is an integer representing the length of the substring. Update your code to look like the sample below.

```
      writer.write(text1, 6, 6);
      writer.newLine();
      writer.write(text2, 0, 4);
```

The first `write` method starts at the sixth character (the `t`) and writes the next six characters (`there` plus the newline character) to the file. The second `write` method starts at character zero (the `m`) and writes the next four characters (`my f`) to the file.

Open practice2.txt

challenge

Try these variations:

Be sure to open the `practice2.txt` file after the change.
* Change the fist `write` method to `writer.write(text1, 12, 1);`

▼ **Why is there an error?**
At first glance, it looks like you are telling Java to print the last character in `text1` which is the newline character. However, you get an error message. The sum of the integers **cannot** be longer than the length of the string. `text1` has a length of 12 and 12 + 1 is 13. If you want to print just the newline character use `writer.write(text1, 12, 0);`

Open practice2.txt

Appending to a File

Appending to a File

You may have noticed that the `write` method completely writes over a file with the new string. If you want to add to an existing file, you need to tell Java that you want to append to a file rather than overwrite it. Start by writing a string to the file `practice3.txt`.

```java
//add code below this line
String path = "studentFolder/text/practice3.txt";
try {
  BufferedWriter writer = new BufferedWriter(new
    FileWriter(path));
  String text = "Nothing left to do";
  writer.write(text);
  writer.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished writing to a file.");
}
//add code above this line
```

Open practice3.txt

To append to this file, use the boolean `true` after the file path when declaring a `FileWriter` object. Set `text` to a different string. **Remember**, Java will append the new text at the end of the file. If you want a space or a new line between the old and new text, you need to add it. We want the new text to appear on its own line, so use the `newLine` method before appending `text` to the file.

```
//add code below this line
String path = "studentFolder/text/practice3.txt";
try {
  BufferedWriter writer = new BufferedWriter(new
    FileWriter(path, true));
  String text = "but smile, smile, smile";
  writer.newLine();
  writer.write(text);
  writer.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished writing to a file.");
}
//add code above this line
```

Open practice3.txt