Learning Objectives - Reading

- **Demonstrate how to open a file in a `BufferedReader` object**
- **Explain what happens when you read from a file that does not exist**
- **Iterate through a file by checking for `null`**
- **Iterate through a file by using the `ready` method**
- **Define the term “token”**
- **Tokenize a string read from a file**
- **Read from one file and write to another**
- **Skip characters in a file with the `skip` method**

Reading a File

Reading a File

Reading a file is similar, in some ways, to writing to a file. You still import the `java.io` package, you need a file path, and you are going to use a `try...catch` block for IO exceptions. Instead of `BufferedWriter` and `FileWriter`, you are going to use the `BufferedReader` and `FileReader` classes.

The `readLine` method will read from the file until it encounters a newline character. All of the text read up until the newline character will be returned. Combine the `readLine` method with the the `println` command to see the first line of the file.

```java
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  System.out.println(reader.readLine());
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

challenge

Try these variations:

- Add another `readLine` method to the program:

```
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  System.out.println(reader.readLine());
  System.out.println(reader.readLine());
  reader.close();
}
```

- Change the `readLine` method to `read`:

```
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  System.out.println(reader.read());
  reader.close();
}
```

▼ **Why does Java print a number?**

The `read` method only reads one character at a time from the file, and it returns the integer value of the character. You can see the character representation of this integer if you change the code to:

```
System.out.println((char)(reader.read()));
```

- Change the file path to:

```
String path = "studentFolder/text/readPractice2.txt";
```

▼ **Why does Java print an IO exception?**
The file `readPractice2.txt` does not exist. Unlike writing to files, Java does not create a new file when reading a file that does not exist.

File Iteration

File Iteration

The `readLine` method only returns one line of a text file. If you want to read an entire file, you will need to iterate over the file, reading each line until you reach the end. Since you do not know how many lines of text a file has, a while loop is preferable. Java represents the end of the file when the `readLine` method returns `null`.

Create string `currentLine` and read a line from the file. Construct a while loop that asks if `currentLine` is not equal to `null`. If this is true, print `currentLine` and then update the variable by reading another line from the file.

```java
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  String currentLine = reader.readLine();
  while(currentLine != null) {
    System.out.println(currentLine);
    currentLine = reader.readLine();
  }
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

challenge

Try these variations:

- Comment out the last line of the while loop:

```
String currentLine = reader.readLine();
while(currentLine != null) {
  System.out.println(currentLine);
  //currentLine = reader.readLine();
}
```

▼ **Why does Java timeout?**

This is an infinite loop. The varaible `currentLine` represents the first line of the text file. If you do not read another line from the file, then `currentLine` will never be equal `null`.

- Change the while loop to look like this:

```
String currentLine;
while((currentLine = reader.readLine()) != null) {
  System.out.println(currentLine);
}
```

▼ **Why does this loop work?**

The reading of the file takes place as the loop checks to see if `currentLine` is not equal to `null`. This means you do not need to read the file when you create `currentLine`, and you do not have to read again inside the loop. Structuring a while loop like this is more concise, but it is not as clear

The Ready Method

The `BufferedReader` class has the method `ready` which returns a `true` if the file can be read. It returns `false` when there are no more lines in the file. You no longer need the `currentLine` variable. Instead, use the `ready` method in the while loop, and print the the value of the `readLine` method. Not only is this a more concise way of iterating over a file, you do not have to worry about an infinite loop.

```java
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  while(reader.ready()) {
    System.out.println(reader.readLine());
  }
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

Tokens

Tokens

Often times when reading text from a file, you will want to parse or analyze the text. Imagine if you wanted to read a file but only print those words that start with a vowel. After reading the line of text, you would need “break up” the string into individual words. These substrings of the original string are known as tokens. This page goes over two ways to create tokens from a string.

The first way of creating tokens is using the string method `split`. This method takes a delimiter as a parameter. A delimiter is the character used to split the string. The `split` method returns an array of strings. The code below will read just the first line of the file and then split it into words since a space is the delimiter. Finally, use an enhanced loop to iterate through the array and print each token.

```
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  String line = reader.readLine();
  String [] tokens = line.split(" ");
  for (String t: tokens) {
    System.out.println(t);
  }
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

challenge

Try these variations:

- Change the delimiter to the string `"a"`
- Change the delimiter to the string `";"`

▼ **Explaining the Output**
Notice that when the delimiter is `“a”` that the letter “a” no longer appears in the tokens. When the delimiter does not appear in the line of text, then one token (the original line of text) is made.

String Tokenizer

Java has the `StringTokenizer` class which allows you to convert strings into tokens with some added features when compared to the `split` method. You need to import the `java.util` package. When instantiating a `StringTokenizer` object, it requires a string and a delimiter as parameters. The methods `hasMoreTokens` (returns a Boolean) and `nextToken` (returns a string) allow you to iterate over the tokens.

```
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  String line = reader.readLine();
  StringTokenizer tokens = new StringTokenizer(line, " ");
  while (tokens.hasMoreTokens()) {
    System.out.println(tokens.nextToken());
  }
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

To convert the entire file to tokens, you will use two nested loops. The first iterates over each line in the file. Use a while loop and the `ready` method from the previous page to read the file. The second loop transforms the string into tokens using another while loop and the `hasMoreTokens` method.

```
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  while (reader.ready()) {
    String line = reader.readLine();
    StringTokenizer tokens = new StringTokenizer(line, " ");
    while (tokens.hasMoreTokens()) {
      System.out.println(tokens.nextToken());
    }
  }
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

challenge

Try this variation:

- Change the delimiter to `a` and add `true` as a parameter:

```
new StringTokenizer(line, "a", true);
```

▼ **Explaining the Output**

The Boolean `true` means that delimiter will be included with the tokens. The delimiters will be their own token. Delimiters are always excluded from the tokens when using `split`.

Skip Method

Skip Method

The `skip` method takes an integer as a parameter, and causes Java to go to a specific character in the text file. The integer is the index for the text file. So `skip(0)` is the first character of the file, `skip(1)` is the second character, etc. The code below prints out the first line from the file.

```
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  System.out.println(reader.readLine());
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

Now compare the output above with the code below. Java will skip the first 30 characters and then print the remaining characters from the first line.

```
//add code below this line
String path = "studentFolder/text/readPractice.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  reader.skip(30);
  System.out.println(reader.readLine());
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

challenge

Try these variations:

- Change the skip amount to 40: `reader.skip(40);`
- Change the skip amount to 400: `reader.skip(400);`

▼ **Why does Java print null?**
The text file only has 227 characters. Skipping 400 characters means there is no more file to read. So Java returns `null`.

Read and Write - Two Files

Read and Write - Two Files

The following code sample reads from one file and writes to another. Since there are two files involved, you will need a `readPath` and a `writePath`. Create a `BufferedReader` with `readPath`, and create a `BufferedWriter` with `writePath`. Using the `ready` method, iterate through the `reader` object. Read a line and then write it to the `writer` object. Do not forget to close both files after the loop terminates. Open `practice4.txt` to make sure that the file has some text.

```java
//add code below this line
String readPath = "studentFolder/text/readPractice.txt";
String writePath = "studentFolder/text/practice4.txt";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(readPath));
  BufferedWriter writer = new BufferedWriter(new
    FileWriter(writePath));
  while(reader.ready()) {
    writer.write(reader.readLine());
  }
  reader.close();
  writer.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a file.");
}
//add code above this line
```

Open practice4.txt

challenge

Try these variations:

Be sure to open the `practice4.txt` file after the change.
* Change the `write` method so that it includes a newline character:

```
writer.write(reader.readLine() + "\n");
```

- Change the `write` method so that it writes the string as uppercase letters:

```
writer.write(reader.readLine().toUpperCase() + "\n");
```

Open practice4.txt