File I/O (Reading)

The purpose of this exercise is to provide you with the opportunity to create command-line applicatior and analyze files.

Learning objectives

After completing this exercise, you'll understand:

- How to read data from a text file.
- How to create complex, interactive command-line applications that are data-driven.
- How to handle file paths provided as application input.
- How to read, interpret, and resolve errors related to file I/O.

Evaluation criteria and functional requirements

- The project must not have any build errors.
- The application returns the expected results.
- The unit tests pass as expected.
  - Note: Tests are only provided for the WordSearch exercise.
- Paths to the input files aren't hard-coded—in other words, the user must be able to enter the path file.
- Your code must be able to handle exceptions for I/O issues, like a missing or unreadable input file.

Getting started

1. Open the `file-io-part1-exercise` project in IntelliJ.
2. Open the Java file for the exercise you're working on. The files are in the `src/main/java/com/teche` package.
3. Provide enough code to get the program started.
4. Verify your work on the command line.
5. Repeat until you've implemented all required features and all unit tests pass.

Part One: WordSearch program

In this exercise, you'll write a program that searches the contents of a file for a word. For each occurren in the file, you'll display the line number and the contents of that line in the console.

You can use any text file on your computer in the program, or you can use the included file called `ali` project folder.

NOTE: Line numbers begin with 1.

The tests for this exercise are in the file `src/test/java/com/techelevator/WordSearchTest.java`. If you r before working on the exercise, all tests fail. As you complete each step, more tests pass.

Explore the starting code

The main logic for `WordSearch` is in the `run()` method. It has a series of prompts for input from the u must provide the following information:

1. The full path of the file to read and search through.

- The user is re-prompted if the file isn't valid. You'll write the logic for checking if the file is valid in

2. The word or words to search for.

- The user is re-prompted if the search term is blank.

3. If search is case-sensitive or case-insensitive.
4. If the output is to contain the line numbers or not.

After the program collects that information, it's passed to the `getMatchingLines()` method where the search logic goes. You'll write the logic for this method as well.

Step One: Checking if the file is valid

When the program prompts the user for a valid file, it calls the method `isFileValid()` with the user p The purpose of the method is to determine if the file is valid before continuing with the program.

Fill in the logic for `isFileValid()`. You must check if the path provided exists and that it's a file (not a Return `true` if the path meets both conditions, otherwise return `false`.

When you complete this step correctly, the `isFileValid_tests` test passes.

Step Two: Read and search the file

For this and the following steps, you'll complete the logic for the `getMatchingLines()` method.

The methods takes four parameters. Refer to the Javadoc comment of the method for the purpose of e
and the return type.

To begin, read the file and return each line that contains the search term.

When you complete this step correctly, the `getMatchingLines_caseSensitiveSearchTerms` test passes.

Step Three: Handle case-insensitive searches

There's a `boolean` parameter that indicates if the search is case-sensitive or case-insensitive. If the valu
search is case-insensitive. That means, if the search term "drink" matches "Drink" and "DRINK"—wherea
sensitive search doesn't.

For this step, add handling of case-insensitive searches and return lines that match without regard to u
lowercase.

When you complete this step correctly, the `getMatchingLines_caseInsensitiveSearchTerms` test passes

Step Four: Add line numbers to output

There's a `boolean` parameter that indicates the output must include the line numbers where each sear
appears.

If the parameter is `true`, add the line number, the character `)`, and a space before the line of text.

For example, if the line number is 23 and the text is "the quick brown fox jumps over the lazy dog", the
be:

```
23) the quick brown fox jumps over the lazy dog
```

When you complete this step correctly, the `getMatchingLines_lineNumbers` test passes.

Step Five: Add exception handling

If you haven't already done so, you need to add handling for any exceptions that may occur during the
process.

Refer to the Javadoc comment of the method for the expected return type of the method. If that excep
`throw` it so the calling code can handle it.

When you complete this step correctly, the `getMatchingLines_ioError_throwsException` test passes.

Part Two: QuizMaker program (Challenge)

object

User Interface Obj

Create a quiz maker program that asks the user a question. You must prompt the user with multiple-ch and allow the user to enter their answer.

object

The program must read the questions from an input file during startup. The questions and answers in t pipe-delimited ("|"), and an asterisk ("*") marks the correct answer.

Log obj.

static?

For example:

```
Question-1|answer-1|answer-2|correct-answer*|answer-4
```

May Change Interface

An example of the file might look something like this:

```
What color is the sky?|Yellow|Blue*|Green|Red
What are Cleveland's odds of winning a championship?|Not likely*|Highly likely|Fat chance|Who
```

Tips

- Use the `.split(String regex)` method to parse file input. The `.split` method divides a string ba provided regular expression and returns the pieces as a `String[]` .
    - A regular expression, or regex, is a specially formatted string that defines a search pattern. Re expressions use some characters for special meanings, including `|` .
    - To use a special character's literal value in Java you put a `\\` in front of it, so `.split("\\|")` file input on each instance of a `|` pipe character.
- Create a class that can hold a quiz question, its available answers, and the correct answer.
- Try holding each quiz question in a list of quiz questions.

Step One: Ask user for input file and display first question

Ask the user a single question when the application starts. Don't show the asterisk in the list of choices

Example

```
Enter the fully qualified name of the file to read in for quiz questions
[path-to-quiz-file]
What color is the sky?
1. Yellow
2. Blue
3. Green
4. Red

Your answer: 2
RIGHT!
```

Step Two: Ask user remaining questions and record correct answers

Go through all of the available quiz questions and ask the user each one sequentially. Record how mar
got correct and print out the total at the end.

Example

```
What color is the sky?
1. Yellow
2. Blue
3. Green
4. Red

Your answer: 1
WRONG!

What are the Cleveland Browns' odds of winning a championship?
1. Not likely
2. Highly likely
3. Fat chance
4. Who cares??

Your answer: 1
RIGHT!

You got 1 answer(s) correct out of the 2 questions asked.
```

Tips and tricks

Practice creating command-line applications

Command-line applications can be a valuable asset to the teams that you work on. You may need to co
routine tasks with files at some scheduled intervals.

Learning how to create applications that can load data into systems, provide alerts when data is incorre automate other systems is an essential skill for developers to gain.

What tasks can you automate in your own life by applying the knowledge you now have about reading files?

Get the file path from the user

Why might it be important to allow people to pass their own file path to the applications you write? Hc you get the path without explicitly asking your customer to provide the path?

Learn how to read error messages

Learning how to read errors that occur when your applications fail is an important skill to develop as ar developer. When an error occurs, reading the error details is a great way to start the discovery process the problems that occurred.

Read the documentation on File I/O

For more information on File I/O in Java, check out the Java File Class API Docs.