File I/O (Reading)

The purpose of this exercise is to provide you with the opportunity to create command-line applications that can read 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 to the input 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-exercises` project in IntelliJ.
2. Open the Java file for the exercise you're working on. The files are in the `src/main/java/com/techelevator` 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 occurrence of the word in the file, you'll display the line number and the contents of that line in the console.

NOTE: Line numbers begin with 1.

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

Processing user input

A pre-defined `Scanner` object to read user input is available in each class of the exercise. It's best practice to use one `Scanner` for reading user input in Java applications.

Before the `main` method, you'll see `private final Scanner userInput = new Scanner(System.in);`. You can use this `Scanner` in your `try` statement like this:

```
try (userInput) {
  // your code that reads user input from the keyboard
}
```

Step One: Collect user input and perform search

When the program runs, it must first prompt the user for a filesystem path and a word to search for in the file.

A file called `alices_adventures_in_wonderland.txt` is in your project folder that you can use as test input. After collecting the required information from the user, you can search through the file for all occurrences of the search word and output that to the console.

For each matching line, your output must contain the line number and the text of the matching line, separated by one or more spaces.

Optionally, you may also use one of the following separator characters between the line number and the text of the matching line, as long as there is some space before the line text:

```
)
:
.
-
```

While reading the file, be sure to handle any I/O exceptions that may occur. Display a message to inform the user about the error. *Remember that it's never a good idea to display the raw exception message to the user.*

Here's an example of what your application could look like:

```
What is the fully qualified name of the file that should be searched?
[path-to-the-file]
What is the search word you are looking for?
dog
604) conversation. "Are you--are you fond--of--of dogs?" The Mouse did not
605) answer, so Alice went on eagerly: "There is such a nice little dog near
617) won't talk about cats or dogs either, if you don't like them!"
622) history, and you'll understand why it is I hate cats and dogs."
1728) "To begin with," said the Cat, "a dog's not mad. You grant that?"
1732) "Well, then," the Cat went on, "you see a dog growls when it's angry,
```

When you complete this part, the `caseSensitiveSearch_LineNumbers` and `caseSensitiveSearch_LineText` tests now pass.

Step Two: Modify program for case-insensitive word search

Modify the WordSearch program to ask the user if they want the search to be case-sensitive.

Here's an example:

```
What is the fully qualified name of the file that should be searched?
[path-to-the-file]
What is the search word you are looking for?
Who
Should the search be case sensitive? (Y\N)
Y
204)           Who stole the Tarts?                          140
518) shall only look up and say, 'Who am I then? Tell me that first, and
1017) "An arm, you goose! Who ever saw one that size? Why, it fills the whole
1042) fancy--Who's to go down the chimney?--Nay, _I_ sha'n't! _You_ do
1169) "Who are _you_?" said the Caterpillar.
1200) "You!" said the Caterpillar contemptuously. "Who are _you_?"
1452) open place, with a little house in it about four feet high. "Whoever
2038) "Who's making personal remarks now?" the Hatter asked triumphantly.
2204) at her, and the Queen said severely, "Who is this?" She said it to the
2371) "Who _are_ you talking to?" said the King, coming up to Alice, and
3042)           Who for such dainties would not stoop?
3050)           "Beautiful Soup! Who cares for fish,
3052)           Who would not give all else for two
3080) [Sidenote: _Who Stole the Tarts?_]
3115) [Illustration: _Who stole the tarts?_]
3435) "Who is it directed to?" said one of the jurymen.
3567) "Who cares for _you_?" said Alice (she had grown to her full size by
```

When you complete this part, the `caseInsensitiveSearch_LineNumbers` and `caseInsensitiveSearch_LineText` tests pass.

Part Two: QuizMaker program (Challenge)

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

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

For example:

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

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 cares??
```

Tips

- Use the `.split(String regex)` method to parse file input. The `.split` method divides a string based on the 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. Regular 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("\\|")` divides your 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 many answers they 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 complete routine tasks with files at some scheduled intervals.

Learning how to create applications that can load data into systems, provide alerts when data is incorrect, or even 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 data from 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? How else might 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 an application developer. When an error occurs, reading the error details is a great way to start the discovery process for diagnosing 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.