**Assignment: TechJobs (Console Edition)**

**Introduction**

Congratulations! Based on your hard work and strong coding skills, you've been brought on as an apprentice at LaunchCode. You're an apprentice member of the LaunchCode Tech Team, and you've been paired with a mentor to help you get comfortable and continue learning.

The Company Team at LaunchCode works with employer partners to match qualified programmers with apprenticeships. They've asked for a new tool to be built to help them easily manage data for currently available jobs. Over the next few weeks, you'll help them build this application alongside mentors from the Tech Team.

This first project will be a simple proof-of-concept prototype. It won't be pretty or have lots of features, but it'll give you a chance to work through some initial concepts and get feedback from LaunchCode staff.

Your mentor on this project is Kathy.

**Learning Objectives**

In this project, you'll show that you can:

* Read and understand code written by others
* Use core Java syntax (methods, variables, loops, conditionals)
* Utilize ArrayList and HashMap collection types
* Work with console I/O via the Scanner class
* Work with data types and arrays

**TechJobs (Console Edition)**

The app you'll be working on is a simple console (i.e. command-line) prototype of the new TechJobs app. It will allow LaunchCode staff to browse and search listings of open jobs by employer partners.

The prototype process will give everybody a chance to work out some initial ideas without investing a ton of time into developing a finished product. Once everybody is happy with the prototype, the Tech Team will begin work toward a full-fledged application.

**Your Assignment**

Kathy has created a console application and started to fill in some features. Her code allows users to search job listings by one of several fields. It also is capable of displaying lists of all of the values of a given field in the system (e.g. all employers, or all locations).

Kathy has handed it off to you with the task of adding a couple of features and then getting feedback from the Company Team.

After you work through the tasks Kathy has laid out for you, tackle one or more of the [bonus missions](https://education.launchcode.org/skills-back-end-java/assignments/techjobs-console/#bonus-missions).

**Getting Started**

Set up a local copy of the project:

* Visit the [repository page](https://github.com/LaunchCodeEducation/techjobs-console-java) for this project and fork the repository to create a copy under your own GitHub account.
* Open IntelliJ (if IntelliJ is currently open, save your work, close it, and reopen it.)
  + If the app opens up to an existing project, select *IntelliJ > Preferences > Appearance & Behavior > System Settings* and uncheck *Reopen last project on startup*. Close and Reopen IntelliJ
* From the "Welcome to IntelliJ" dialog box, select *Check out from Version Control > GitHub*
* Choose your fork from the repository dropdown, select the parent directory where you'd like to store your project, and hit *Clone*.
* When asked "Would you like to create an IDEA project..." select *Yes*, and then accept all of the default options that are presented.
* In the screens that follow, be sure to choose *Create Project From Existing Sources* on the first pane, and select the default values of all following panes.

Before diving in and starting to code, make sure you understand what the code you've been given does. Since you're starting with a functioning - albeit unfinished - program, go ahead and run it to get an idea of how it works. To do this, right-click on the main method in the TechJobs class and select *Run TechJobs.main()*.

The application will run until you force it to quit, re-prompting time after time. To kill it, press the red "stop" icon in the Run pane. We'll learn precisely how the program manages to work this way below.

Let's explore the code by starting with the source of the data our program is providing access to.

**The Data File: jobs\_data.csv**

Our simple app doesn't connect to a database. If the prototype proves useful and we continue development, we'll add that functionality later. But for now, we've been given a CSV (comma-separated values) file from the Company Team at LaunchCode that contains some recent jobs. This file was exported from an Excel spreadsheet into this format, which is easy for programs to read in.

If CSV files are new to you, don't worry, they're easy to understand. CSV files are conceptually similar to simple spreadsheets in that they organize data in rows and columns. The major difference is that they don't have the ability to carry out calculations the way spreadsheets do, and you can easily open, read, and edit them in plain text editors.

Open up jobs\_data.csv, which is in the resources folder at the top level of the project. You'll see that the first line is:

name,employer,location,position type,core competency

While it isn't required, the first line of a CSV file often represents the column names. We have 5 names here, which indicates that each of our rows in the CSV file should have 5 fields. In this file format, a "row" corresponds to a new line. So each line below the first will constitute a row of data, or a record.

Have a look a the data below line 1, and ask yourself the following questions:

* Which fields match up with which column names above?
* Why do some lines/rows (e.g. line 10) have more commas than others, if commas are supposed to separate columns?
* What role do the double-quotes play?

**The TechJobs Class**

The TechJobs class contains the main method that will drive our program's functionality. It contains three methods:

1. main - The main application runner.
2. getUserSelection - A utility method that displays a menu of choices and returns the user's selection.
3. printJobs - This is meant to print a list of jobs to the console in a nicely formatted manner, but hasn't been implemented yet. This will be part of your job.

Let's look at each of these.

**The main method**

The logic within main presents menus in turn, and based on the user's choice, takes appropriate action.

It begins by declaring two local variables: columnChoices and actionChoices. These contain information relating to the menus that we'll display, and we'll look at them in more detail later.

Next, we notice a while loop that starts while (true). This may seem odd, but actually, it makes a lot of sense after a short explanation. We want our application to continually run until the user has decided they want to quit. The simplest way to do this is to loop forever. When the user wants to quit, they can kill our program by pressing ctrl-C (a widely-known command to kill a console application). As you saw above, however, IntelliJ's Run pane works slightly differently and you'll need to rely on the red "stop" icon to stop the program.

The main method can be summarized as follows:

1. Present the user with choices on how to view data: list or search.
2. Based on that choice, prompt them with the column that they would like to apply the choice to. In the case of a search, we also ask for a search term.
3. Carry out the "query" to the JobData class via one of its public methods.
4. Display the results of the "query."
5. Repeat.

The word "query" is in quotes here because we're not really carrying out a database query, but the net effect is the same as if we were. We ask a method for data that originates from a non-Java source, it parses and filters that data, and gives it back to us.

**The getUserSelection method**

The getUserSelection method takes in a String to display above the menu, to provide context for what they are being asked. It also takes in a HashMap with String keys and String values. How is this used? What will this HashMap contain when the method runs?

To figure this out, right-click on the method name and select *Find Usages*. This will open a pane and display each location in the program where getUserSelection is called. The first such usage is the first line of the main while loop:

`String actionChoice = getUserSelection("View jobs by:", actionChoices);`

What is this HashMap named actionChoices? If we look a few lines above, we see:

// Top-level menu options

HashMap<String, String> actionChoices = new HashMap<>();

actionChoices.put("search", "Search");

actionChoices.put("list", "List");

If you recall how the program worked when you ran it, the first menu that you chose had two options, Search and List, which seem to correspond to the entries in actionChoices. This is, in fact, the case. This is the data that is used to generate the first menu we see when running the program.

The second usage of getUserSelection is a few lines below:

String columnChoice = getUserSelection("List", columnChoices);

This references columnChoices, which is declared at the top of main and has a similar structure to actionChoices (they're the same data type and are used in calls to the same method, so this shouldn't be surprising). Most of the entries in columnChoices correspond to columns in the jobs data set, but there's one additional entry with key/value pair "all"/"All". These entries will help us present to the user the options for searching our data, which will correspond to searching within a given column, or searching all columns at once.

The keys in actionChoices and columnChoices represent the "internal" String we'll use to refer to these options -- for example, when representing the user's menu choice, or querying data -- and the values in the map represent the "external" way that these are represented to the user.

Within getUserSelection itself, most of the code is within a do-while loop. A [do-while loop](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html) is similar to a while loop, but the conditional check is at the *end* of the loop's code block. This has the net consequence that the loop's code block *always runs at least once*. At the end of the block's execution, we check a condition to determine if we should run the block again. This nicely mimics the behavior of simple menu-driven applications.

Within this loop, menu options are printed to the screen and user input is collected. If the input is valid, it returns the choice as a String to the caller. This String corresponds to the chosen key (from choices, which will be either actionChoices or columnChoices) of the item the user selected. If invalid, it re-prompts the user.

The local variable choiceKeys is used to easily enumerate the choices HashMap. In other words, it gives us a simple way to provide an ordering to choices, which doesn't have an ordering of its own.

**The JobData Class**

The JobData class is responsible for importing the data from the CSV file and parsing it into a Java-friendly format, that is, into HashMap and ArrayList form. Look toward the bottom of the class and you will see a method named loadData, which does just what it advertises. After parsing the file data, it stores the data in the private property allJobs which is of type ArrayList<HashMap<String, String>>.

We haven't covered static properties and methods in-depth yet. For this assignment, know simply that they allow us to use properties and methods of a class without creating an object from that class. For example, we can call JobData.findAll() from the TechJob class.

If you want to create a new method in JobData, or add a property, be sure to declare it static.

Let's look more closely at the data type of allJobs. It purports to be an ArrayList that stores HashMap objects which have String keys and String values. If we were to represent some of this data visually, using [] for an ArrayList and {} with key/value pairs (as in Python lists and dictionaries), it would look like this:

[

{

"name": "Junior Data Analyst",

"employer": "Lockerdome",

"location": "Saint Louis",

"position type": "Data Scientist / Business Intelligence",

"core competency": "Statistical Analysis"

},

{

"name": "Junior Web Developer",

"employer": "Cozy",

"location": "Portland",

"position type": "Web - Back End",

"core competency": "Ruby"

},

...

]

If you look at loadData you'll see a lot of unfamiliar code. Kathy wrote this essential piece of code for you, and while you won't have to modify it, it will be useful to have an idea of how it works. Read through the code until you feel like you can describe its functionality at a basic level.

There are three more methods in JobData, each of which is public (and static, per our earlier note): findAll(), findAll(String), and findByColumnAndValue(String, String). Note that there are two methods named findAll, but this is allowed in Java via a feature called **overloading**. Overloading happens when multiple methods have the same name, but they each have different input parameters (also called argument lists). Read more about [overloading](http://beginnersbook.com/2013/05/method-overloading/).

Here are a few questions to ask yourself while reading this code:

* What is the data type of a "job" record?
* Why does findAll(String) return something of type ArrayList<String> while findByColumnAndValue(String, String) and findAll() return something of type ArrayList<HashMap<String, String>>?
* Why is loadData() called at the top of each of these four methods? Does this mean that we load the data from the CSV file each time one of them is called?

**Your Tasks**

Here are the tasks for you to carry out for your first apprenticeship assignment.

**Implement printJobs**

When trying out the program, and later when reading the code, you hopefully noticed that there's some work to do in the printJobs method. As it stands, it currently just prints a message: "printJobs is not implemented yet".

Implement this method. It should print out something like this:

\*\*\*\*\*

position type: Data Scientist / Business Intelligence

name: Sr. IT Analyst (Data/BI)

employer: Bull Moose Industries

location: Saint Louis

core competency: Statistical Analysis

\*\*\*\*\*

If there are no results, it should print an appropriate message.

To do this, you'll need to iterate over an ArrayList of jobs. Each job is itself a HashMap. While you can get each of the items out of the HashMap using the known keys ("employer", "location", etc), think instead about creating a nested loop to loop over each HashMap. If a new field is added to the job records, this approach will print out the new field without any updates to printJobs.

Test this method before moving on to your next step:

1. Save your changes
2. Select *Run* from the Run menu and choose to run the TechJobs class (or if you have recently run it, just select the green arrow in the top right corner of the screen)
3. Select "1" to list the jobs, and then "0" to list them all
4. Make sure the printout matches the styling above
5. Test that it prints a descriptive message if no jobs are found by selecting "0" to search and then "3" to search for a location. Then enter a location that is not in the data (e.g., "Cancun"). Your message should be displayed

**Create Method findByValue**

At this stage, the application will allow users to search a *given column* of the data for a given String. Your next task is to enable a search that looks for the search term in *all* of the columns.

In the JobData class, create a new (public static) method that will search for a String within each of the columns. Name it findByValue. Here are a few observations:

1. The method that you write should not contain duplicate jobs. So, for example, if a listing has position type "Web - Front End" and name "Front end web dev" then searching for "web" should not include the listing twice.
2. As with printJobs, you should write your code in a way that if a new column is added to the data, your code will automatically search the new column as well.
3. You *should not* write code that calls findByColumnAndValue once for each column. Rather, utilize loops and collection methods as you did above.
4. You *should*, on the other hand, read and understand findByColumnAndValue, since your code will look similar in some ways.

You'll need to call findByValue from somewhere in main. We'll leave it up to you to find where. You might have noticed that when you try to search all columns using the app, a message is printed, so that is a good clue to help you find where to place this new method call. Once you find where to call your new method, you can *Run* the program again to test your code.

**Make Search Methods Case-Insensitive**

You've completed your first two tasks! Then you demoed the updated application or the Company Team and they noticed a feature that could be improved. When searching for jobs with the skill "JavaScript" some results were missing (e.g. the Watchtower Security job on line 31 of the CSV file). The search methods turn out to be case-sensitive, so they treat "JavaScript" and "Javascript" as different Strings.

The Company Team has *demanded* (ahem, *strongly requested*, they politely clarify) that this needs to be fixed. And you've told them that you're up to the task.

Here are some questions to ask yourself as you get started:

* Which methods are called when searching?
* How is the user's search String compared against the values of fields of the job HashMap objects?
* How can you make this comparison in a way that effectively ignores the case of the Strings?
* How can you do this *without* altering the capitalization of the items in allJobs, that is, so that you don't change the data, and consequently it is printed out the same way that it appears in job\_data.csv?

You might find it useful to review the String methods listed in the chapter on [Data Types](https://education.launchcode.org/skills-back-end-java/java4python/data-types/#string).

When this task is completed, you're done!

**Sanity Check**

Before submitting, make sure that your application:

* Prints each field of a job when using search functionality, and when listing all columns. If there are no search results, a descriptive message is displayed.
* Allows the user to search for a String across all columns.
* Returns case-insensitive results.

**Solution Demo**

Watch a demo of a working solution.

**How to Submit**

To turn in your assignment and get credit, follow the [submission instructions](https://education.launchcode.org/skills-back-end-java/assignments/).

**Bonus Missions**

If you want to take your learning a few steps further, here are some additional problems you can try to solve. We're not providing you much guidance here, but we have confidence that you can figure these problems out!

* **Sorting list results**: When a user asks for a list of employers, locations, position types, etc it would be nice if results were sorted alphabetically. Make this happen.
* **Returning a copy of allJobs**: Look at JobData.findAll(). Notice that it's returning the allJobs property, which is a static property of the JobData class. In general, this is not a great thing to do, since the person calling our findAll method could then mess with the data that allJobs contains. Fix this by creating a copy of allJobs. (*Hint:* Look at the constructors in the Oracle ArrayList documentation.)