COS125 - Precept 9 (IO)

Download `precept9.zip` from the precepts webpage, unzip and open the project folder. The folder contains two files with personal and healthcare information on (fictitious) residents of Massachussets, `healthcare_data.txt` and `personal_data.txt`. They are formatted similarly to a CSV file, with three modifications for your convenience:

1) Whitespace characters inside entries were removed;
2) Empty fields are identified by the character `*`; and
3) Fields are separated by a space (instead of a comma).

1 Search

Write a program `Search.java` that takes in a string search term from the command line, reads a file from standard input, and then outputs all lines with a value that matches the search term using `s.equals(t)`.

Note that in order to print an entire entry, you need to use a `StdIn` method that reads all tokens until the next line (`StdIn.readString()` only reads until the next whitespace). You may also find the `String` function `split()` useful.

Try out your program on a single search term, such as the age *71*:

```
java-introcs Search 71 < healthcare_data.txt
```

Now string together multiple searches using pipes to answer the following:

1) How many people are age 71 and female? `___`
2) How many people are age 71, male, and asian? `___`
3) How many people are age 25, female, and hispanic? `___`
4) How many people are white, hispanic, female, and 13 years old? `___`

2 Counting Matches

The previous program was helpful for displaying entire records for people who matched the given characteristics. However, the program will struggle to search files where different features can share the same values. For instance, in `healthcare_data.txt`, gender is easily searchable because it is the only characteristic with "F" or "M" as options. That is not true in `personal_data.txt`, where "M" is used for gender and marriage status.

Write a program `CountMatches.java` that reads a pair of strings from the command line, a characteristic (column name) and the value to search for, and counts the number of entries in the database that are exactly equal to it.

Use this program to count the following in `personal_data.txt`:

1) How many people are married men? `___`
2) How many people are married women? `___`

2.1

Now count the number of matches for the strings in the table above (top row is age, bottom row is zip code) using `healthcare_data.txt` and whichever of your created program you prefer.

3	4	28	30	34	41	51	71	73	90	101
01060	01105	01604	01835	01841	01940	02138	02148	02149	02641	02726

Test your code by comparing the results with those you obtained in the previous precept!

3 Music with MIDI

Create a program `PlayWithMidi.java` that receives two command-line arguments – a `String` instrument name and an `int` tempo – sets the instrument and tempo accordingly and plays the melody in `StdIn` where each note is defined by an `int` MIDI note and a `double` duration.

Your code should interpret the string `piano` as the instrument that `StdMidi` specifies by the constant `ACOUSTIC_GRAND_PIANO`; and `violin` as that corresponding to `VIOLIN`.

```
> java-introcs PlayWithMidi piano 100
60 1.0 60 1.0 67 1.0 67 1.0 69 1.0 69 1.0 67 2.0 65 1.0 65 1.0 64 1.0 64 1.0 62
1.0 62 1.0 60 2.0
```

4 Rock Paper Scissors

After the conditionals lecture, we created a program that simulated a single run of rock paper scissors using a command line argument for the user input. Modify the included `RockPaperScissors.java` code in this precept to make the game instead take in a single odd-numbered integer representing total rounds to play. Then run that many rounds of rock paper scissors, allowing the user to make a choice at the start of each round using standard input. The first side to win `totalRounds/2 + 1` rounds, wins. Make sure ties are re-run.

5 Database Search (Bonus)

Copy `CountMatches.java` into `DatabaseSearch.java`, and adapt it so it reads an arbitrary number of feature-value pairs of strings. Then it should read either included data file of your choice from `StdInt` and prints the entries whose corresponding features match all such values.

Note that the outputs of the two commands below should be exactly the same. Use this to test your own single vs. multi feature searching program.

```
> java DatabaseSearch GENDER [gender] < healthcare_data.txt | java DatabaseSearch
ZIP [zip]
> java DatabaseSearch GENDER [gender] ZIP [zip] < healthcare_data.txt
```