Joanna Klukowska
joannakl@cs.nyu.edu

Project 2: San Francisco Restaurant Ratings

Due date: September 28, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will provide a tool for searching through the data set of San Francisco restaurnts' inspection results.

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- working with multi-file programs
- reading data from input files
- using and understanding command line arguments
- working with large data sets
- using the `ArrayList` class
- writing classes
- working with existing code
- extending existing classes (inheritance)
- parsing data

Most, if not all, of the skills that you need to complete this project are based on the material covered in cs101. But there might be certain topics for which you did not have to write a program or that you forgot. Make sure to ask questions during recitations, in class and on Piazza.

Start early! This project may not seem like much coding, but debugging always takes time.

For the purpose of grading, your project should be in the package called `project2`. This means that each of your submitted source code files should start with a line:
`package project2;`
Keep in mind that spelling and capitalization are important!

Dataset

In this project you will be working with open data. Wikipedia has a good description of open data: "Open data is the idea that some data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control."

The data set that you need can be found at DataSF portal. For your convenience a csv file is also located on the course website. It is a listing of restaurant inspection results for restaurants in San Francisco starting from mid-2015. The data set is provided by San Francisco Department of Public Health.

The file that you download is a CSV (comma separated values) file - it is a simple text file and is processed like text file (but it can also be opened by most of the spreadsheet programs and displayed column-wise based on the locations of commas on each line). Since some of the columns contain longer text that might, optionally, contain commas as well, those entries are enclosed in a set of double quotes.

There are seventeen columns in the dataset. Some of the columns may be empty.

From the point of view of this assignment, a valid row in the data set MUST contain the business_name, business_postal_code, and inspection_date. Any rows that do not contain all of those fields should be quietly ignored by the program.

Joanna Klukowska
joannakl@cs.nyu.edu

User Interface

Your program has to be a console based program (no graphical interface) - this means that the program should not open any windows or dialogs to prompt user for the input.

Program Usage

The program is started from the command line (or run within an IDE). It expects one command line argument.

This program should use command line arguments. When the user runs the program, they provide the name of the input file as a command line argument. (This way the user can specify a data set from another city or a different version of the data set).

The user may start the program from the command line or run it within an IDE like Eclipse - **from the point of view of your program this does not matter**.

If the name of the input file provided as a command line argument is incorrect or the file cannot be opened for any reason, the program should display an error message and terminate. It should not prompt the user for an alternative name of the file.

If the program is run without any arguments, the program should display an error message and terminate. It should not prompt the user for the name of the file.

The error messages should be specific and informative, for example:

```
Error: the file restaurant_scores.csv cannot be opened.
```

or

```
Usage Error: the program expects file name as an argument.
```

The above error messages generated by your code should be written to the `System.err` stream (not the `System.out` stream).[1]

Input and Output

The program should run in a loop that allows the user to issue different queries. The two types of queries are:

```
name KEYWORD
```

and

```
zip KEYWORD
```

In the first case, the program should display the list of all the restaurants in the data set whose name contains the given keyword (this search should be case insensitive) with two most recent dates of scored inspection results[2]. The user may enter a multi-word KEYWORD. The matching restaurants should be displayed in alphabetical order.

In the second case, the program should display the list of three restaurants with the highest scores on the most recent scored inspection in the zip code (postal code) specified by the keyword. The restaurants should be displayed from the highest score to the lowest score. (If there are fewer than three restaurants in the given zip code, all the restaurants should be displayed without duplicates.)

On each iteration, the user should be prompted to enter a new query (for which the program computes the results) or the word 'quit' to indicate the termination of the program.

The user should not be prompted for any other response.

If the query entered by the user is invalid, the program should display an error message:

```
This is not a valid query. Try again.
```

and allow the user to provide an alternative query.

If the query entered by the user does not return any results, the program should print a message

```
No matches found. Try again.
```

[1] If you are not sure what the difference is, research it or ask questions.

[2] A *scored inspection* is an inspection for which a numerical score was assigned. For some dates, there are multiple inspection records - often with the same score. The program should display the information about two most recent dates - this can be two or more actual inspection records.

and allow the user to provide an alternative query.

Matching reults display: If the query entered by the user matches one or more restaurant, the information about all the matching restaurants should be displayed in the following format:

```
RESTAURANT_NAME
------------------------------------
address                 : ADDRESS
zip                     : ZIP
phone                   : PHONE_NUMBER
recent inspection results:
    SCORE_1, DATE_1, RISK_CATEGORY_1, VIOLATION_DESCRIPTION_1
    SCORE_2, DATE_2, RISK_CATEGORY_2, VIOLATION_DESCRIPTION_2
```

All the words in uppercase letters are place-holders for the actual values from the data set. If any of the values is missing, it should be left blank and the corresponding commas should not be shown. The inspection results should be shown for the two most recent inspections that had a numerical score (i.e., the inspections without an actual score assigned should not be shown in the string).

Sample interaction:

Sample user interaction is shown in Appendix 2.

Data Storage and Organization

You need to provide an implementation of several classes that store the data and compute the results when the program is executed.

In particular, your program must implement and use the following classes. You may implement additional classes and additional methods in the required classes, if you wish.

`Date` Class

The `Date` class is used to represent dates. It should store the numerical values for month, day and year. The class should provide two constructors:

```
Date( String date )
Date (int month, int day, int year)
```

The first constructor should throw an instance of `IllegalArgumentException` if it is called with a `null` parameter, or a string that does not match either `MM/DD/YYYY` or `MM/DD/YY` pattern, or a string that matches the pattern, but for which the values are invalid (see below). Note: the `MM` and `DD` specifications should always be exactly two characters long, so for January 5, 2019, the string will be ”01/05/19” or ”01/05/2019”.

The second constructor should throw an instance of `IllegalArgumentException` if it is called with invalid numerical values. The valid values for month are 1-12 (inclusive). The valid values for year are 2000 to 2025 (inclusive). The lower bound for valid values for day is always 1 and the upper bound depends on the month and year:

- 31 for January, March, May, July, August, October and December
- 30 for April, June, September and November
- 29 for February in leap years: 2000, 2004, 2008, 2012, 2016, 2020, 2024
- 28 for February in all other years

There should not be a default constructor.

The class should implement `Comparable<Date>` interface. The result of comparing two `Date` objects should be that the earlier date is smaller and the later date is larger.

The class should override the `toString()` method. It should return a string in the format ”MM/DD/YYYY”.

Joanna Klukowska
joannakl@cs.nyu.edu

Note about date column in the data set.
The inspection date in the data set uses two different formats: "MM/DD/YY HH:MM:SS XM" or "MM/DD/YYYY HH:MM:SS XM". The time shown is always midnight and you can ignore it. But the year is specified either as a four digit or a two digit value. The `Date` class constructor should be able to handle either of those. The `toString()` method should always produce the four digit representation for the year.

`Inspection` Class

The `Inspection` class is used to represent the particular inspection of a restaurant. It should store the date of the inspection, the assigned score, the violation description and the risk category. The class should provide a four parameter constructor

```
public Inspection (Date date, int score, String violation, String risk)
```

The constructor should throw an instance of `IllegalArgumentException` if

- it is called with `null` value for `date`;
- it is called with the score outside of the valid range of 0 to 100 (inclusive).

The two string parameters are allowed to be `null` or empty (this will happen when there is no violation description and/or risk assesments for the given inspection, i.e., the corresponding column in the data set is empty).

`Restaurant` Class

The `Restaurant` class is used to represent restaurants. It should store the following information:

- name of business
- address of business
- phone number
- zip code / postal code
- list of inspections

Note: most of the restaurants appear on multiple rows of the data set since there are more than one inspection records for them.

This class should provide two constructors:

```
public Restaurant(String name, String zip)
public Restaurant(String name, String zip, String address, String phone)
```

The following describes acceptable values for each of the constructor parameters. If a constructor is called with an invalid parameter, it should throw an instance of `IllegalArgumentException`.

- `name` - any non-empty string
- `zip` - a string containing exactly five digits
- `address` - any non-empty string to indicate the address, `null` or an empty string when the data is not available (for example, when the column in the data set is empty)
- `phone` - any string to indicate the phone number, `null` or an empty string when the data is not available

There should be no default constructor.[3]

The class should provide a method

```
public void addInspection(Inspection inspect)
```

that adds a given inspection to the list of inspections for the current `Restaurant` object. The method should throw an instance of IllegalArgumentException if it is called with `null` parameter.

This class should implement `Comparable<Restaurant>` interface. The comparison should be done by the `name` of business as the primary key (case insensitive, alphabetical ordering - this is the ordering provided by the `String` object comparison function), and by the `zip` as the secondary key (i.e., when two objects that have the same value of `name` are compared, the comparison should

[3] A default constructor is one that can be used without passing any arguments.

be performed by `zip`) (the zip codes should be ordered from smaller numerical value to larger - this is the ordering provided by the `String` object comparison function).

This class should override the `equals` methods. The two `Restaurant` objects should be considered equal, if their `name` and `zip` are identical (the name comparison should be case insensitive).

This class should override the `toString` method. It should return a string matching the following format:

```
RESTAURANT_NAME
------------------------------------
address              : ADDRESS
zip                  : ZIP
phone                : PHONE_NUMBER
recent inspection results:
    SCORE_1, DATE_1, RISK_CATEGORY_1, VIOLATION_DESCRIPTION_1
    SCORE_2, DATE_2, RISK_CATEGORY_2, VIOLATION_DESCRIPTION_2
```

All the words in uppercase letters are place-holders for the actual values from the data set. If any of the values is missing, it should be left blank and the corresponding commas should not be shown. The inspection results should be shown for the two most recent inspections that had a numerical score (i.e., the inspections without an actual score assigned should not be shown in the string).

The class should provide getters and setters for any data fields that your program may need access to from the outside of the class.

`RestaurantList` Class

The `RestaurantList` class should be used to store all the `Restaurant` objects. This class should inherit from the `ArrayList< Restaurant>` class.

The class needs to provide a default constructor that creates an empty `RestaurantList` object.

The class should implement the following two methods:

```
    public RestaurantList getMatchingRestaurants ( String keyword )
    public RestaurantList getMatchingZip ( String keyword )
```

The first method should return a list of `Restaurant` objects whose names contain the `keyword` as a substring (case insensitive). The second method should return a list of `Restaurant` object whose zip codes are equal to the `keyword`. The returned lists should be sorted according to the natural order[4] of their elements. These methods should return `null` if the functions are called with a keyword that is either equal to `null` or an empty string. They should also return `null` if there are no matches for the keyword.

The class should override the `toString` method. This method should return a `String` containing a semi-colon and space separated list of the names of the `Restaurant` objects stored in the list. For example, if the three `Restaurant` objects stored in the list arre *AFC Sushi @ Safeway #1206*, *Dong Hing, LLC* and *Fior d' Italia* , than the returned `String` should be

`"AFC Sushi @ Safeway #1206; Dong Hing, LLC; Fior d' Italia"`.

You may implement other methods, if you wish.

`SFRestaurantData` Class

The `SFRestaurantData` class is the actual program. This is the class that should contain the `main` method. It is responsible for opening and reading the data files, obtaining user input, performing some data validation and handling all errors that may occur (in particular, it should handle any exceptions thrown by your other classes and terminate gracefully, if need be, with a friendly error message presented to the user[5]).

You may implement other methods in this class to modularize the design.

[4]The natural order of elements is determined by the `compareTo` method defined in the class of the elements.

[5]The program should never just reprint the exception message as a way of handling an exception. These messages are rarely readable to the ordinary user and make it seem like the program crashed in response to the exception.

Joanna Klukowska
joannakl@cs.nyu.edu

Given Code

To simplify parsing of the data from the input file, you can use the function **`splitCSVLine`**. Given a row (a single line) from the input data set, it returns an **`ArrayList<String>`** object containing all the individual entries from that row. This function produces empty strings to represent entries that were blank within the input line before the last comma. If the last columns in the row are missing, the returned **`ArrayList<String>`** object might have fewer entries.

For example: given a string containing the line:

```
79804,CurveBall,428 11th St,San Francisco,CA,94103,,,,14150287297,79804_20190325,03/25/2019 12:00:00 AM,,Structural
 Inspection,,,
```

it returns an **`ArrayList<String>`** object with 16 strings:

```
["79804", "CurveBall", "428 11th St", "San Francisco", "CA" , "94103", "", "", "", "14150287297", "79804_20190325
", "03/25/2019 12:00:00 AM", "", "Structural Inspection", "", ""]
```

and given a string containing the line:

```
97105,New King Dumpling,3319 Balboa St,San Francisco,CA,94116,,,,,97105\_20190102,02/01/19 12:00 AM,83,Routine -
Unscheduled,97105\_20190102\_103119,Inadequate and inaccessible handwashing facilities,Moderate Risk
```

it returns an **`ArrayList<String>`** object with 17 strings:

```
["97105", "New King Dumpling", "3319 Balboa St", "San Francisco", "CA", "94116", "", "", "", "", "97105\_20190102
", "02/01/19 12:00 AM", "83", "Routine - Unscheduled", "97105\_20190102\_103119", "Inadequate and inaccessible handwashi
 facilities", "Moderate Risk"]
```

The source code for **`splitCSVLine`** is given in Appendix 1. You can use this method as is or modify it if you wish. (You should read it and make sure that you understand what it does.)

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at `https://cs.nyu.edu/~joannakl/cs102_f19/notes/CodeConventions.pdf`.

The data file should be read only once! Your program needs to store the data in memory resident data structures.

You may not use any of the collection classes that were not covered in cs101 (for this assignment, do not use **`LinkedList`**, **`Stack`**, **`Queue`**, **`PriorityQueue`**, or any classes implementing the **`Map`** interface). You can, and should, use the **`ArrayList`** class.

You may use any exception-related classes.

You may use any classes to handle the file I/O, but probably the simplest ones are **`File`** and **`Scanner`** classes. You are responsible for knowing how to use the classes that you select.

Working on This Assignment

You should start right away!

You should modularize your design so that you can test it regularly: for example, implement the part of the code that reads and parses the input file, then implement and test individual classes, then implement the part that provides the interactive part of the program, ...

Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should make sure that you are testing the program on much smaller data set for which you can determine the correct output manually. You can create a test input file that contains only a few rows.

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

Each class that you submit will be tested by itself without the context of other classes that you are implementing for this assign-

ment. This means that you need to make sure that your methods can perform their tasks correctly even if they are executed in situations that would not arise in the context of this specific program.

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, push it to a private git repository, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens.

Grading

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment. Make sure that you are submitting functioning code, even if it is not a complete implementation so that you can get partial credit.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing and how to work with it.

40 points class correctness: correct behavior of methods of the required classes and correct behavior of the program

30 points design and the implementation of the required classes and any additional classes

20 points proper documentation, program style and format of submission

10 points academic honesty questionnaire - take in the form of an online quiz

How and What to Submit

For the purpose of grading, your project must be be in the package called `project2`. This means that each of your submitted source code files should start with a line:
`package project2;`

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to Gradescope.

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select ”Save in zip format”, ”Compress the contents of the file” and ”Create only selected directories”
- click Finish

You may resubmit to Gradescope as many times as you wish before the submission link closes.

For this project, you will see ALL of the results for the autograded unit tests. This will not be the case in future assignments.

Joanna Klukowska
joannakl@cs.nyu.edu

Appendix 1: Parsing CSV file

```java
/**
 * Splits the given line of a CSV file according to commas and double quotes
 * (double quotes are used to surround multi-word entries so that they may contain commas)
 * @author Joanna Klukowska
 * @param textLine  a line of text to be passed
 * @return an Arraylist object containing all individual entries found on that line
 */
public static ArrayList<String> splitCSVLine(String textLine){

if (textLine == null ) return null;

  ArrayList<String> entries = new ArrayList<String>();
  int lineLength = textLine.length();
  StringBuffer nextWord = new StringBuffer();
  char nextChar;
  boolean insideQuotes = false;
  boolean insideEntry= false;

  // iterate over all characters in the textLine
  for (int i = 0; i < lineLength; i++) {
    nextChar = textLine.charAt(i);

    // handle smart quotes as well as regular quotes
    if (nextChar == ’"’ || nextChar == ’\u201C’ || nextChar ==’\u201D’) {

      // change insideQuotes flag when nextChar is a quote
      if (insideQuotes) {
        insideQuotes = false;
        insideEntry = false;
      }else {
        insideQuotes = true;
        insideEntry = true;
      }
    } else if (Character.isWhitespace(nextChar)) {
      if ( insideQuotes || insideEntry ) {
      // add it to the current entry
        nextWord.append( nextChar );
      }else { // skip all spaces between entries
        continue;
      }
    } else if ( nextChar == ’,’) {
      if (insideQuotes){ // comma inside an entry
        nextWord.append(nextChar);
      } else { // end of entry found
        insideEntry = false;
        entries.add(nextWord.toString());
        nextWord = new StringBuffer();
      }
    } else {
      // add all other characters to the nextWord
      nextWord.append(nextChar);
      insideEntry = true;
    }

  }
  // add the last word ( assuming not empty )
  // trim the white space before adding to the list
  if (!nextWord.toString().equals("")) {
    entries.add(nextWord.toString().trim());
  }

  return entries;
}
```

Joanna Klukowska
joannakl@cs.nyu.edu

Appendix 2: Sample Interaction

Note that some of the text is wider than the shown display. Your program does not need to worry about line-wrapping.

```
Search the database by matching keywords to titles or actor names.
  To search for matching restaurant names, enter
    name KEYWORD
  To search for restaurants in a zip code, enter
    zip KEYWORD
  To finish the program, enter
    quit


Enter your search query:

name nopa
Nopa
------------------------------------
address              : 560 Divisadero St
zip                  : 94117
phone                :
recent inspection results:
93, 09/04/2018, High Risk, Unclean or unsanitary food contact surfaces

Nopa Liquors & Deli
------------------------------------
address              : 1601 Fulton St
zip                  : 94117
phone                :
recent inspection results:
100, 06/25/2018

Nopalito
------------------------------------
address              : 306 Broderick St
zip                  : 94117
phone                :
recent inspection results:
91, 04/19/2019, Low Risk, Unapproved or unmaintained equipment or utensils
91, 04/19/2019, High Risk, Unclean or unsanitary food contact surfaces
90, 04/23/2018, Moderate Risk, Moderate risk food holding temperature
90, 04/23/2018, Low Risk, Unapproved or unmaintained equipment or utensils
90, 04/23/2018, Moderate Risk, Inadequately cleaned or sanitized food contact surfaces

Nopalito
------------------------------------
address              : 1224 9th Ave
zip                  : 94122
phone                : 14155430303
recent inspection results:
87, 01/16/2019, Low Risk, Low risk vermin infestation
87, 01/16/2019, Moderate Risk, Moderate risk food holding temperature
87, 01/16/2019, High Risk, Improper cooling methods
90, 02/14/2018, Moderate Risk, Employee eating or smoking
90, 02/14/2018, Low Risk, Unclean or degraded floors walls or ceilings
90, 02/14/2018, Low Risk, Low risk vermin infestation
90, 02/14/2018, Low Risk, Improper food storage



Enter your search query:

name clif
THE CLIFT ROYAL SONESTA HOTEL
------------------------------------
address              : 495 GEARY ST
zip                  : 94102
phone                :
recent inspection results:
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
Enter your search query:

zip 94109
PINE & JONES MARKET
------------------------------------
address              : 1100 PINE ST
zip                  : 94109
phone                : 14155242876
recent inspection results:
100, 12/04/2019

Crostini and Java
------------------------------------
address              : 899 Hyde St
zip                  : 94109
phone                :
recent inspection results:
100, 08/22/2019

Starbucks #6927
------------------------------------
address              : 685 Beach St
zip                  : 94109
phone                :
recent inspection results:
100, 07/05/2019



Enter your search query:

find 94109
This is not a valid query. Try again.


Enter your search query:

name zxzxzx
No matches found. Try again.




Enter your search query:

quit
```