**CSE 174 – Sections A-H – Fall 2016 PROGRAM #5: 35 points –**

**Due Sunday, October 2, by 11:59 p.m.**

**Outcomes:**

* Write programs that manipulate String objects
* Research API documentation to find useful methods
* Document test cases

**Scoring:**

* If you do not submit a zip file containing your source code, your score will be zero.
* If you submit source code that does not compile, your score will be zero.
* Deductions will be made for not meeting the usual requirements:
* Source code that is not formatted according to the usual guidelines
* File and class names that do not meet specifications

|  |  |  |
| --- | --- | --- |
|  | **Full credit** | **No credit or Partial credit** |
| **Write a solution that works for the given test case**  **(4 points)** | Your solution works for the test case that was given in this assignment. | Your solution does not work for the given test case. |
| **Write a solution that works for all other test cases**  **(20 points)** | Your solution works for all test cases that the user attempts. | Your solution does not work for some or all of the other test cases. |
| **Show the test cases you used in your comments**  **(6 points)** | The top of your comments include a list of names, phone numbers, and addresses you tested, indicating the results from each | You did not include the test cases, or your test cases did not adequately cover special cases. |
| **Organize your solution in a way that is very readable and understandable**  **(5 points)** | Your logic, variable names, and comments make it easy for others to understand your solution. | Your solution is difficult to follow (unclear variable names, lines of code that attempt to do too much, and so on). |

**Getting started:**

1. Review String objects and methods from chapter 2 of your text.
2. Look up the Java API documentation for the String class, and familiarize yourself with other useful String methods.
3. Use Dr. Java's "Interactions Pane" to experiment with String methods. For example, what happens when you type "mississippi".indexOf("i", 2) and why does it happen?

**Sample run of program5:**

|  |
| --- |
| Name: **SMITH , Mary jean**  Phone: **513-529-1809**  Address: **204 Main Street, Apt. 5, Oxford, Ohio 45056**  \*\*\*\*\* Results \*\*\*\*\*  Mary J. Smith  (513)529-1809  204 Main Street, Apt. 5  Oxford, Ohio 45056 |

**Requirements:**

Your program should read the name, address, and phone information from a user, and then "parse" that information to display it in a different format. Your program should run as shown in the sample run above. Some specifics:

1. Prompt the user for her name, address, and phone information, as shown in the sample run, matching that format exactly. The text shown in bold is not text that is printed by your program. Rather, it is the text entered by the user.

Note that Scanners have a method named nextLine(). Use this method to obtain each of the three lines of user input. This takes the entire line entered by the user and stores it in the variable. For example:

String origAddress = keyboard.nextLine();

1. Write code that breaks apart each of the user-entered pieces into smaller, more useful pieces. For example, you should extract the user's first name and last name into separate variables. This is the part where you will do the most problem solving. Be organized. It can be useful to declare extra variables to hold intermediate results, but too many extra variables can sometimes make it harder to keep track of things. Name variables in a way that would be meaningful to other programmers who are trying to understand your code.
   * Bad names: lastName1, lastName2, lastName3
   * Better names: lastNameFirstLetter, lastNameAfterFirstLetter, lastNameFinal

Sometimes it is helpful to *reuse* a variable name. For example, instead of this:

lastNameLowercase = lastName.toLowerCase(); // Uses 2 variables

Try this instead:

lastName = lastName.toLowerCase(); // Uses 1 variable

1. Write code that prints out a "cleaned up" version of the user-entered information, formatted as shown in the sample run. More specifically:

* The first line of output is always \*\*\*\*\* RESULTS \*\*\*\*\*
* The second line of output is the first name, followed by the middle initial, followed by the last name. Only the first letter of each name and the initial should be uppercase.
* The third line of output is the phone number, formatted as shown in the example.
* The fourth line of output is the street address. This is everything entered by the user in the address line, except for the city, state, and zip code.
* The final line of output is the city, state, and zip code, as entered by the user in the address line.
* The output should contain no extra spaces at the beginning or end of each line.

**Assumptions:**

To make this program somewhat more manageable, you should assume that the information entered by the user will conform to certain guidelines, as outlined below:

**Name:**

* The user will always enter her last name, followed by a comma and space, followed by her first name, followed by a space, followed by her middle name.
* There will always be at least one space after the comma, and there will be at least one space between the first and middle names.
* There may be extra spaces separating the pieces (as shown in the example above), but there will never be extra spaces IN a piece. In other words, if a person's last name is Smith, there may be several spaces before or after Smith, but the user *won't* enter

Sm i th

* The name may contain an odd mix of lowercase and uppercase letters, but when it gets printed, the uppercase and lowercase letters must be "cleaned up" (first letter of each part is uppercase, and the rest lowercase).

**Phone:**

* The phone number will always consist of 3 main parts: a 3-digit area code, followed by a hyphen, followed by a 3-digit exchange, followed by a hyphen, followed by the last 4 digits.
* There may be extra spaces anywhere within the phone number. All of these should be eliminated when it is time to print the phone number. So, for example, if the user enters:

5 1 3 -5 2 9 - 180 9

your program should still print:

(513)529-1809

**Address:**

Pay careful attention to commas: There will always be exactly one comma separating the street address from the city, and exactly one comma separating the city from the state, but the street address itself may contain many more commas (such as the comma before "Boulevard" in the example above.

* Assume that the uppercase and lowercase letters are entered correctly for the address line, and do not need to be changed. In other words, the Name may have had strange upper and lower case, but the address never will.
* Assume that the spacing of the address line is entered correctly, and does not need to be changed. In other words, the Name may have had extra spaces, but the address never will.
* Note that the sample contained an apartment number. That is NOT something that will always happen. Read carefully the requirements of the Address part of the code.

**Testing:**

The document you are reading contains only one test case. Your program should work with any name, address, and phone number (any that fit the assumptions above). Part of your assignment is to create your own set of carefully chosen test cases, and to document those test cases in the comments of your code. You should think about the parts of the user input that could be particularly challenging (for example, spaces at the beginning of the user input, spaces in the middle of the user input, input that is all uppercase, input that is all lowercase, and so on).

In the comments of your code at the top, include 3 sections listing names you tested, phone numbers you tested, and addresses you tested. Include the actual results from each test. For example, you might have this in your code:

// Names and results:

// SMITH , Mary jean ---> Mary J. Smith

// Smith, Mary Jean ---> Mary J. Smith

The goal is not to have the most test cases, but to have carefully selected test cases. For example, if you want to test how your program handles names entered in all uppercase, you only need to test that once. It is possible to have a lot of test cases, but forget to test an important aspect of the assignment.

**Submitting:**

Follow these steps to submit your work:

* 1. Create an empty folder named program5
  2. Put your source code file (.java) in the program5 folder.
  3. There should be no other files in the program5 folder
  4. Compress the folder itself to create a zip file. Name the zip file program5.zip
  5. Submit only the zip file to the Canvas website.

Note: If you submit your work and decide to modify the file, you need to resubmit a new zip file containing the updated source code. Do not rename your source code file. Do not rename the folder. Canvas may add a number to the name of your zip file. That is fine. But you should keep all filenames and folder name the same.