CSE114 Spring 2023 : Assignment 2

Due: Mar 21, 2023 at 11:59 PM [KST]

Read This Right Away

For the due date of this assignment, don't go by the due date that you see on Brightspace unless you have set your timezone to KST. By default, Brightspace shows times in EST/EDT in the US.

Directions

- At the top of every file you submit, include the following information in a comment
  - Your first and last name
  - Your Stony Brook email address
- Please read carefully and follow the directions exactly for each problem.
- Your files, Java classes, and Java methods must be named as stated in the directions (including capitalization). Work that does not meet the specifications may not be graded.
- Your source code is expected to compile and run without errors. Source code that does not compile will have a heavy deduction (i.e. at least 50% off).
- You should create your program using a text editor (e.g. emacs, vim, atom, notepad++, etc).
- Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.

What Java Features to Use

For this assignment, you are *not* allowed to use more advanced features in Java than what we have studied ***at the time I post the assignment to Brightspace***. If you have a question about features you may use, please ask!

What to Submit

Combine all your .java files from the problems below into a single zip file named as indicated in the **Submission Instructions** section at the end of this assignment. Upload this file to **Brightspace** as described in that section.

Multiple submissions are allowed before the due date. Only the last submission will be graded.

Please do **not** submit `.class` files or any that I did not ask for.

Partial vs. Complete Solutions

Your programs should compile and run without errors, both compile-time and run-time errors! Please do not submit programs that do *not* even compile! Its better to submit a partial solution that compiles and runs as opposed to an almost complete solution that does not even compile. A program that does not compile even if it is very close to being perfect would only receive less than 50% maximum of the possible score for the program! This policy applies not only to this problem set but also to all the other ones in the remainder of the semester. I will not repeat this in each problem set though. So, please remember this.

Naming Conventions In Java And Programming Style In General

Please use these conventions as you establish your *programming style.* Programming professionals use these, too.

- **Names:** Choose informative names,
  - e.g. `hourlyRate` rather than `hr` for a variable name.
  - `CheckingAccount` rather than `CA` for a Java class name.
- **Class names:** We start a class name with an uppercase letter as I have in my examples: `Hello` not `hello` or `HELLO`. For multi-word names you should capitalize the first letter of each word, This is called 'Pascal' case.
  - e.g. `UndergraduateStudent`, `GraduateStudent`, `CheckingAccount`
- **Variable names:** We start a variable name with a lowercase letter. This is called 'Camel' case.
  - e.g. `count`, `hourlyRate`, averageMonthlyCost
- **Method names:** Naming convention for method names is the same as that for variable names.
- **Use of white space:** Adopt a good indentation style using white spaces and be consistent throughout to make your program more readable. Most text editors (like emacs and vim) should do the indentation automatically for you. If this is not the case, see me and I can help you configure your setup.

I will not repeat these directions in each assignment, but you should follow this style throughout the semester.

Problem 1 (10 Points)

Write a Java class named **Volumes** in a file named **Volumes.java** that adds two sets of volume measurements as follows. For example, you can add two volumes: 1 gallon, 2 quarts 1 cup and 2 gallons, 3 quarts, 1 cup. The resulting sum should show 4 gallons, 1 quart, 2 cups.

Use three sets of variables. Each set has a variable to hold gallons, a variable to hold quarts and a variable to hold cups. Two sets of variables are for the volumes input from the user. The third set of variables will hold the sum of the other two sets (so there will be a variable each for gallons, quarts, and cups). So add the volumes from the first two sets of variables and store the result in the third set of variables.

Finally, print the three volume values with annotations so the result will be meaningful to the user. If you need to introduce additional variables, feel free to do so.

Read the values for the inputs with a Scanner. The user should provide values for gallons, quarts, and cups for each of the two volumes. All values entered by the user should be integers.

Example:
Volume 1:
Enter gallons (int): 1
Enter quarts (int): 2
Enter cups (int): 1
Volume 2:
Enter gallons (int): 2
Enter quarts (int): 3
Enter cups (int): 1
Totals: Gallons: 4 Quarts: 1 Cups: 2

Hand in **Volumes.java.**

Problem 2 (10 points)

Write a Java class named **RollingAvg** in a file named **RollingAvg.java** that does the following. It has seven integer variables declared with the names n1, n2, n3, n4, n5, n6, and n7. Your program should assign an integer to each of them at compile time. The numbers that you use are up to you. Now, your program must calculate their average and store it to a variable named **average** of type double.

As output, the program should print each of the seven integers separated by spaces. On the next line, print the sum of the 7 numbers. Finally, on a third line, print the average of the numbers.

Average of n1 through n7 is ##.##

Where ##.## is the average (as a 'double' value) of the 7 integers. Use System.out.printf () and limit the size after the decimal point to 4 digits.

Now, you will recompute the average one step at a time. You will average the first two numbers (n1 and n2). Print the average as follows:

Average of n1 through n2 is ##.##

Where ##.## is the average you just computed. Let's call that average ravg (rolling average).

Now calculate the average of the first 3 values but do not start with n1, n2, and n3. Instead, you will average the result of the last calculation along with n3. This is a 'weighted' average. The formula is:

The new average value is:

ravg = ((2 * ravg) + n3) / 3.0;

(Note: 2 is the 'previous value count' averaged. The variable n3 is the 'next value' and the denominator (3.0) is the 'new value count'.

Print that average as you did for n1 and n2.

Each succeeding rolling average is computation is:

ravg = ((previousvaluecount * ravg) + n#) / (double) newvaluecount;

After you reach and print the average for n1 through n7 (using the rolling averages calculation), that value should match your original calculation at the start of the program. This technique (rolling averages) is used when you don't know the number of values or the values themselves ahead of time. Instead, you just read another number (from the console or from some sensor) and you 'roll it' into the existing average.

Example output

```
===============
java RollingAvg
The values are: 5 10 7 50 33 21 2
The sum is: 128
Average of n1 through n7 is    18.2857
Average of n1 through n2 is     7.5000
Average of n1 through n3 is     7.3333
Average of n1 through n4 is    18.0000
Average of n1 through n5 is    21.0000
Average of n1 through n6 is    21.0000
Average of n1 through n7 is    18.2857
```

Hand in RollingAvg.java.

Problem 3 (5 Points)

Write a Java class named **Change** in a file named **Change.java** that reads from the keyboard a double value representing a monetary amount (in US dollars). Then, determine the fewest number of each bill and coin needed to represent that amount, starting with the highest (assume that a 50 dollar bill is the largest bill available). The bill and coin denominations in circulation are:

$50 bill
$20 bill
$10 bill
$5 bill
$1 bill [=100 cents]
Half Dollar coin [50 cents]
Quarter [25 cents]
Dime [10 cents]
Nickel [5 cents]
Penny [1 cent] (but use the plural form 'pennies')

For example, if the value entered is 77.83, then the program should print the equivalent amount as:

```
1 $50 bills
1 $20 bills
0 $10 bills
1 $5 bills
2 $1 bills
1 Half dollar coin
1 quarters
0 dimes
1 nickels
3 pennies
```

Note that you do not need to worry about the singular/plural form of the words notes and coins. Just use the plural form. Also note that you can (and should) solve this without any conditionals (if statements) since we learn those in an upcoming lecture. It can be solved simply with mathematical operations!

Hand in **Change.java**.

Submission Instructions

Please follow this procedure for submission:

1. Place the deliverable source files (`Volumes.java, RollingAvg.java, Change.java`) into a folder by themselves. The folder's name should be CSE114_PS2_<yourname>_<yourid>. So if your name is Alice Kim and your id is 12345678, the folder should be named 'CSE114_PS2_AliceKim_12345678.

2. Compress the folder and submit the zip file.
    a. On windows, do this by pressing the right-mouse button while hovering over the folder. Select 'Send to -> Compressed (zipped) folder'. The compressed folder will have the same name with a .zip extension. You will upload that file to the Brightspace dropbox for **Assignment2.**
    b. On mac, move the mouse over the folder then right-click (or for single button mouse, use Control-click) and select **Compress**. There should now be a file with the same name and a .zip extension. You will upload that file to the Brightspace assignment for **Assignment2.**
3. Navigate to the course Brightspace site. Click **Assignments** in the left column menu. Click **Assignment2** in the content area. Under **ASSIGNMENT SUBMISSION**, click **Browse My Computer** next to **Attach Files.** Find the zip file and click it to upload it to the web site.