Data Structures and Algorithms, Spring ’23 section 1 Dr. Hristescu
Lab 1 (IOOP & OOPDA Review Lab)

Assigned: Thursday, January 19
Due: **10:00pm Monday, January 23**
Submit Problems 1, 2, 3 and for extra credit Problem 4 (for ALL: code+sample runs on ALL provided input data- all code formatted with astyle), status and conclusions.

Any difficulty in solving these review problems may indicate that you do not have the necessary background for this course. If this is the case, please make an appointment with me ASAP.

Step 0: Login to your elvis account. Elvis is now running a Fedora 37 (Linux) system.
0.1 If don’t already have it installed, google “MobaXterm download”, find it and install the portable version on your computer
0.2 Run it and open an SSH “New session” using for Basic SSH settings:
Remote host: elvis.rowan.edu and **Port:** 22
You will be asked to provide your regular username and password.
0.3 Check the content of your feedback file using:
```
~hristescu/DSA/Labs/Process/feedback
```

Step 1: Create a directory (folder) called DSA for this course.
1.1 At the prompt type in the make directory command
```
mkdir  DSA
```

1.2 Check that the directory is there by listing the content of the directory. At the prompt type in the command
```
ls
```

1.3 Change directory to your newly created directory, typing in the command
```
cd   DSA
```

1.4 At the prompt type use the make directory command to create a directory for this lab. Call it Lab1.
```
mkdir  Lab1
```

2.1 Change directory to your newly created directory, typing in the command
```
cd   Lab1
```

Step 2: Familiarize yourself with one of the two editors **vi(vim)** or **emacs.**
vi: Quick Introduction to the vi editor (my favorite editor) or the improved version
vim Quick Introduction to the vim editor
emacs: Quick Introduction to the emacs editor Comprehensive Emacs Manual

vim is a very simple editor, emacs is more sophisticated and offers a friendlier interface.

Step 3: Copy and run a Hello Class program in your Lab1 directory.
3.1 Copy the file called HelloClass.java from the class web site into your Lab1 directory using the following command:
```
cp  ~hristescu/public_html/classes/dsa/HelloClass.java  .
```

3.2 Look at the content of your directory to see the file using the command:
```
ls
```

3.3 Look at the content of the file in your directory using the command:
```
more HelloClass.java
```

3.4 Compile the HelloClass program using the command
```
javac HelloClass.java
```

3.5 Check the content of your directory using the ls command. The java compiler should have created a file called HelloClass.class in the current (Lab1) directory. This file contains the program in bytecode format which is ready to run. Run the program using the command:

```
java HelloClass
```
The output of this program should be: **Hello Class**

Step 4: Copy and run a more sophisticated Hello Class program in your Lab1 directory.

4.1 Copy the file called DrHsHelloClass.java from the class web site into your Lab1 directory using the following command:

```
cp ~hristescu/public_html/classes/dsa/DrHsHelloClass.java .
```

4.2 Look at the content of your directory to see the file using the command:

```
ls
```

4.3 Look at the content of the file in your directory using the command:

```
more DrHsHelloClass.java
```

4.4 Compile the HelloClass program using the command

```
javac DrHsHelloClass.java
```

4.5 Run the program using the command:

```
java DrHsHelloClass
```

The program will ask you to enter a name. Type in a name. For instance, if you type in **Dr. Hristescu**, the output of this program should be:

```
Dr. Hristescu shouts Hello Class!!!
```

Note: The implementation had to:

- create an InputStreamReader using the standard input stream
- create a BufferedReader called stdin using the InputStreamReader created (static –why?)
- use the BufferedReader to read a line of text provided by the user as input using the readLine method
- add the throws IOException clause to the header of any method that calls the readLine method.

Problem 1: Create and run an even more sophisticated Hello Class program in your Lab1 directory.
Design and edit a file called MyHelloClass.java with the following content using your favorite editor. Make sure that the class name and the file name match!!!
For instance, if you chose the vi(m) editor, at the prompt type in:

```
vi Lab1P1Driver.java
```

Your program should:

- ask the user to enter an integer number **n**
- read **n** names provided by the user
- displays the provided names followed by "**shout(s) Hello Class!!!**" using correct grammar and punctuation.

Here are three sample runs of this program:

Sample run 1:

```
elvis prompt > java Lab1P1Driver
Enter number of people:1
Enter name number 1:Joe
Joe shouts Hello Class!!!
```

Sample run 2:

```
elvis prompt > java Lab1P1Driver
Enter number of people:2
Enter name number 1:John
Enter name number 2:Jane
John and Jane shout Hello Class!!!
```

Sample run 3:

```
elvis prompt > java Lab1P1Driver
Enter number of people:3
Enter name number 1:John
Enter name number 2:Jane
Enter name number 3:Joe
John, Jane, and Joe shout Hello Class!!!
```

Sample run 4:

```
elvis prompt > java MyHelloClass
Enter number of people:5
Enter name number 1:John
Enter name number 2:Jane
Enter name number 3:Joe
Enter name number 4:Jack
Enter name number 5:Jim
John, Jane, Joe, Jack, and Jim shout Hello Class!!!
```

Note: You will have to read the integer into a String and then parse the string into an integer using the parseInt **class method** for the Integer class (You may want to use the Java Class Libraries Reference)

Step 5: Use the program from Problem 1 with input and output redirection:
5.1 Edit a file called i1_P1.input with the following content:

```
3
John
Jane
Joe
```

5.2 Modify your program by adding after each line that reads data in a line that echoes the read-in data. You will have to use this for **every program** you submit this semester. It is also advisable to prompt the user for input by using System.out.**print**, NOT println as shown below.

Example:
```
System.out.print("Enter name: ");
String name = stdin.readLine();
System.out.println(name);
```

Important: It is **strongly advised** that you **trim** your read-in data (especially if it is of **numeric type**)

5.3 Run the modified program using the following command:

```
java  Lab1P1Driver  <i1_P1.input  >Lab1P1sampleruns.txt
```

5.4 View the content of the file Lab1P1Sampleruns.txt

5.5 Repeat 5.1 – 5.4 for all the additional input data supplied for Problem 1 while appending to **Lab1P1Sampleruns.txt** using the command similar to:

```
java Lab1P1Driver <i2_P1.input  >> Lab1P1Sampleruns.txt
```

Problem 2:

P2.1 Write a program that computes the weighted arithmetic mean for a user specified number of (number,weight) pairs as well as the simple arithmetic mean of the numbers specified:

- ask the user to enter an integer number n for the pairs
- prompt and read **n** pairs (integer number, real number for the weight) provided by the user
- displays the weighted arithmetic mean of all the **n pairs** and the simple arithmetic mean of the **n numbers**

Make sure that:

- if **n** is specified to be **0**, the program computes the correct values. Include a sample run for this input.
- The program works for any n specified (do not hardcode the number of pairs)

Example1:
Input file contains:

```
6
 9
3.4
24
46.6
67
9.5
9
23.1
 84
45
 32
3.1
```

After running your program with the input redirected to the above input file, the redirected output file should contain the following sample run:

```
Enter number of pairs: 6
 Enter number 1: 9
Enter weight 1: 3.4
Enter number 2: 24
Enter weight 2: 46.6
Enter number 3: 67
Enter weight 3: 9.5
Enter number 4: 9
Enter weight 4: 23.1
 Enter number 5: 84
Enter weight 5: 45.
 Enter number 6: 32
Enter weight 6: 3.1
 Simple arithmetic mean of these 6 numbers is: 37.50
 Weighted arithmetic mean of these 6 pairs is: 44.93
```

Example2:
Input file contains:

```
0
```

Output file should contain:

```
Enter number of pairs: 0
 Simple arithmetic mean of these 0 numbers is: 0
 Weighted arithmetic mean of these 0 pairs is: 0
```

P2.2 Add the following **header** updating the required info. All your files from now on should include the header below.

```
/*
 * Purpose: Data Structure and Algorithms Lab X Problem Y [instantiate X and Y]
 * Status: Barely started/ Incomplete/ Complete and thoroughly tested [choose only one!!!]
 * Last update: 09/06/22
 * Submitted:  09/07/22
 * Comment: test suite and sample run attached
 * Comment: I declare that this is entirely my own work
 * @author: John Doe [replace with your own first and last name]
 * @version: 2022.09.06
 */
```

You can either edit with vim by:

- cutting and pasting the lines containing the header above or
- positioning your cursor where you want the lines included and then the following:
  :r ~hristescu/public_html/classes/dsa/header.txt

P2.3 On **Lab1P2Driver.java** run the command:

```
astyle --mode=java Lab1P2Driver.java
```

Examine the file **Lab1P2Driver.java** that should now contain the formatted code. The original is kept in a file with a .orig extension. For more formatting options refer to the astyle manual. **ALL your code from now on should be formatted using astyle.**

Problem 3: **For this problem you will have to review in detail ArrayList, your IOOP material that pertains to processing collections and specific features only available starting with Java 8.**
Write a program that keeps track of a collection of individual characters. Use an **ArrayList** collection as the **ONLY** datafield of your class. Call the datafield **data**.
Add the following functionality:

1. a method called **add** that allows the addition of one new item (character) at the end of the collection
2. for **every** possible different approach of processing an ArrayList collection design methods called **displayNameOfProcessingMethod** that display the content of the collection beginning with the first item and ending with the last item. You should have quite a few display methods! Document them and include a write-up with the number of methods found. Make sure to replace **NameOfProcessingMethod** in the method name with the appropriate descriptive name for the approach used.

3. For every possible approach that you can apply that does not require the use of additional storage, design methods called **displayReverseNameOfProcessingMethod** that display the content of the collection beginning with the last item and ending with the first item. Document how many of the methods you found. How many of the approaches that you used for display could be used for displayReverse?
4. For every possible approach that you can apply that does not require the use of additional storage, design methods called **displayEveryThirdNameOfProcessingMethod** that display every third item in the collection starting with the first item - i.e displays items in positions 1, 4, 7, 10, 13, 16, 19, etc (if they exist).
5. For every possible approach that you can apply that does not require the use of additional storage, design methods called **testIfPalindromeNameOfProcessingMethod** that returns true if the collection is a palindrome, false otherwise. Document how many methods you found.

Develop **your own test suite** to thoroughly test your solution and include the sample runs in your submission.
Below is info from ReviewFrame:

Special consideration: ArrayList collection processing using the following approaches:

- direct index access in basic loop (initial language design)
- iterators (Iterator/ListIterator) (Java 2 and later)
- enhanced (foreach) loop (Java 5 and later)

Java 8 (and later) -specific:

- forEach method + lambda expression
- forEach method + method reference

When to use each approach and their limitations. Investigate in the context of **efficiently***** implementing the following tasks on an ArrayList collection:

a. displaying the content of the collection starting with the first item and ending with the last item
b. displaying the content of the collection starting with the last item and ending with the first item
c. checking if the content of the collection is a palindrome

*** **Important efficiency considerations**: no additional storage for the collection, no repeated passes through the collection.

Step6:
Lab1Conclusions: Summarize in a few sentences what you learned from working on this lab.

Step7: (if you have not done this already as part of the pre-semester prep)
7.1 Change to your home directory using the command:

```
cd
```

7.2 Edit the **.bash_profile** file (example **vi .bash_profile**) and add this line by either cutting and pasting it or using **:r ~hristescu/public_html/classes/dsa/bash_lines.txt**
And then replacing "Your Name" with your name and updating the section number.

7.3 Source it with the command:

```
source .bash_profile
```

Future sessions will not require any explicit sourcing, since the sourcing is done automatically upon login. You can also check them using the **alias** command.
To check if makepdf is now in effect run the following command and check is it lists it:

```
compgen -c| grep makepdf
```

Step8:
8.1 Change directory to your Lab1 directory, typing in the command

```
cd   DSA/Lab1
```

8.2 Gather IN THIS ORDER the **status file**, **conclusion file**, all the **source code files,** your **sample run(s) file** for Problems 1 and 2 and 3 using the command:
more Lab1Status.txt Lab1Conclusions.txt Lab1P1Driver.java Lab1P1Sampleruns.txt Lab1P2Driver.java Lab1P2Sampleruns.txt Lab1P3Driver.java Lab1P3Sampleruns.txt > allfiles.txt

where Lab1P1Driver.java (or the name you used for it) is the source file for Problem 1
and **Lab1P1Sampleruns.txt** (or the name you used for it) is the sample run(s) file for Problem 1
and **Lab1Conclusions.txt** contains your conclusions on this Lab from Step 6

Important: What to include in the **allfiles.txt** for this submission and all future submissions:
Include: for all problems the source code, extensive sample runs on ALL provided input data or your own input data if data was not supplied, status and conclusions

DO NOT INCLUDE: any input data, any code that was posted for the lab that you used but have not made any changes to it

8.3 Use the command **makepdf** command that will generate (or overwrite if it already existed) the **submission PDF** file.

makepdf allfiles.txt

8.4 Check to see what the submission PDF file contains. **That is what will be graded.**

Important: It is very important for this as well as any future submission to make sure that allfiles.txt contains ONLY TEXT files (.java , .txt) and **NO** pictures (no .jpg, no .pdf) and no .class files. If any non-text files are included in allfiles.txt, then the PDF file will be corrupted.

Step9:
Follow the instructions to turn in the lab electronically by the deadline.

Step10:
Check the files that you have submitted for Lab1 using:

```
~hristescu/DSA/Labs/Process/what intime
```

Extra Credit (do not attempt unless the regular lab is fully completed)
Problem 4:
Time, plot, compare and analyze the efficiency of all methods developed for Problem 3 above. Start with a large number of randomly generated items in the collection and continue to increase the input size to achieve a more accurate comparison. Use a **command line argument** (**args[0])** to specify the number of items.
P4.1 **Java's System.nanoTime()**. Please note that nanoTime returns the wall clock time. It can be used to measure the elapsed time that depends on the load in the system and therefore does not provide a very accurate measure. In your timing make sure that you do not include the data generation or the actual displaying on the collection. The timing should only include the traversal that collects the information in the StringBuilder.
P4.2 Use the Linux **time** utility on elvis that reports the wall clock, user, and system time for the **entire application**.
P4.3 Are there any other metrics that could be considered that would provide better accuracy?
P4.4 Which one of the two timing methods is the better measure for the comparing these methods and why?
Assemble the timing results, analysis and comparison in a document that you place in the **Lab1** directory that is named **P4** with the appropriate extension (.txt,.doc,.pdf). Do not include it in allfiles.txt unless it is in text format.