**IT 2045C Computer Programming II  
Prof. Tom Wulf   
Fall 2023**Practicum 01 Java File IO and Safe Input review **20 points**

# Learning Goals:

* Review Java text file IO
  + Be able to write programs that read and write java text files
* Review the SafeInput library we developed in CP I. All our console programs will use this library
* Write two suites of programs each with a reader and writer to create and read data files
* Be sure to retain a copy of this and all work for the course. You will need these programs and files later as we work through the labs.
* Note: we are switching to the IntelliJ editor this time. It will be used in future java courses so you should use it. If you see a reference to Netbeans use IntelliJ instead.

# Directions:

Create a GitHub Repo called **Practicum** and make it public. (After you have a grade, you can change it to private.)

**Do not create java packages here. Use the default package. That rule is for all assigned work!**

# Part 1: Person

**Files: PersonGenerator.java, PersonReader.java, SafeInput.java  
  
Note: include and use SafeInput.java in both of these projects. (Be sure to place a separate copy within each of the project folders.) The SafeInput.java source file is available with the assignment directions in Bb, just download and include it in each project. Keep a copy handy with your work for use throughout the course. (We will review in class during the first week how to use SafeInput.)**

1. Create a program (java main class) called **PersonGenerator.java**.
2. Your program will prompt the user to enter lines of data for a file on persons. You don’t know ahead of time how many Persons will be entered so use an ArrayList to store the records.
3. Once the user indicates they have entered all the data elements for each person save it to a text file using a name they provided. (Be sure to only save complete sets of elements.)
4. Be sure to use the SafeInput library to completely bullet-proof your program. Block and repeat until the user gets the input correct. Do not terminate.  
     
   **INSERT SEVERAL SCREEN SHOTS SHOWING THE INPUT OF THE DATA. BE SURE TO GET ONE OF A COMPLETE RECORD.**

A screenshot of a computer program

Description automatically generated  
**Inputting data into the class**

1. Here is the data and file format. One record per line. Each line has all of the data elements (a – e)
   1. ID (a String)
   2. FirstName
   3. LastName
   4. Title (a string like Mr., Mrs., Ms., Dr., etc.)
   5. YearOfBirth (an int)
2. These files contain no headers. There is one person record per line in the comma-delimited format. Here is a sample record:

**000001, Bilbo, Baggins, Esq., 1060**

1. Be sure to test your program carefully.

Use your program to create a data file called **PersonTestData.txt** with these records:  
 **000001, Bilbo, Baggins, Esq., 1060**

**000002, Frodo, Baggins, Esq., 1120**

**000003, Samwise, Gamgee, Esq., 1125**

**000004, Peregrin, Took, Esq., 1126**

**000005, Meridoc, Brandybuck, Esq., 1126**

1. Leave a copy of the data file in the project folder, Please remove other test files.  
     
   **DISPLAY A SCREEN SHOT OF THIS FILE IN THE IntelliJ EDITOR HERE!**

A screenshot of a computer

Description automatically generated

**data.txt in Intellij**

# Part 2: PersonReader.java

1. Now create a new java main class in the same project called **PersonReader.java**.
2. Be sure to use JFileChooser and SafeInput.
3. Create a program that prompts the user to select an existing Person file and then displays the file to the screen.
4. Use your **PersonTestData.txt** file to test and debug your program.

Use String.format to create a neatly formatted columnar display of the data records

**ID# Firstname Lastname Title YOB  
=====================================**   
**000001 Bilbo Baggins Esq. 1060**

1. **Frodo Baggins Esq. 1120**

**…**

**GET SCREENSHOTS OF:  
- THE FILECHOOSER RUNNING**

* **THE DISPLAY OF THE CHOOSEN FILE**

**A screenshot of a computer

Description automatically generated**

**File opener working**

**A screenshot of a computer

Description automatically generated**

**PersonReader displaying the file’s contents**

# Part 3: Product

**As the teletubbies say: “Again!, Again!”**

1. Create the two programs **ProductReader.java** and **ProductWriter.java**
2. Here is the product info:
   1. ID (a String as before in Person)
   2. Name (a String)
   3. Description (a String a short sentence)
   4. Cost (This is currency so it will be a Java double)
3. Use your program to create the following **ProductTestData.txt** file   
     
   **000001, Pipeweed, Long Bottom Leaf, 600.0**

**000002, Lembas, Elven Wayfare Bread, 200.0**

**000003, Wine, Woodland Elf Wine, 400.0**

**000004, Mushrooms, Farmer Took’s Finest, 125.0**

**000005, Mithril, Enchanted Dwarven Armor, 3000.0**

**A screenshot of a computer program

Description automatically generated**

**ProductTestData.txt in IntelliJ**Part 4: ProductReader.java

1. Implement ProductReader to use a formatted display as you did in PersonReader.
2. Remove any other test files but leave a copy of this file in your project.
3. **Include screen shots of your program test output that establish that you completed and tested all the code for the lab at the end of this file. Label the shots so I can easily determine that you did the work. Make sure they are very legible. Follow what I asked for in the first part.  
     
   A screenshot of a computer

   Description automatically generated**

**ProductReader display the file’s contents**

**INCLUDE THE URL OF YOUR PUBLIC GitHub Repo here:** [**https://github.com/fakejoshwash/Practicum01**](https://github.com/fakejoshwash/Practicum01)

**Submission:**

Save this file as **Lastname\_Firstname\_Pract01\_Review.docx** that includes each of the complete IntelliJ project folders. Add this .docx file that you are currently reading with your embedded screen shots into the archive. **DO NOT INLCUDE SEPARATE SCREEN SHOT FILES INSTEAD OF EMBEDDING THEM HERE!**

**Just submit the word.docx**  
  
I expect you to follow the submission directions to the letter. I won’t accept submissions that are not formatted correctly using the naming conventions and formats I have specified. If you make a simple mistake all assignments are set to allow you to resubmit a corrected copy. I’ll only examine the last one you submit. Once a grade is recorded for a valid submission you can not resubmit.