IT 2045C Computer Programming II

Prof. Tom Wulf

Fall 2024 v1.75

20 points

Practicum 01 Java File IO and Safe Input review

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.

Directions:

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

- Create a single project for the assignment. All the java files are in the src directory. You only need a single copy of the SafeInput.java library here. You can then use it in any of the other four programs.

- Please use the specified file names. Remember that GitHub does not allow spaces in project names so we avoid the by using the underscore _.

- Never use the Scanner to read text files **as it is not thread-safe**. Instead we use the Java 1.8 NIO File and Path classes to read text files using BufferedReader and BufferedWriter. This is true throughout the course.

Resources:

SafeInput.java -

This is a library of static input functions that we created in CP I. You place it in the src folder. It allows us to avoid recoding the input bullet-proofing code again and again. We will go over it in class and you can watch the support video.

NIO_Path_File_Complete.zip

This is a zip compressed IntelliJ project file with several short and detailed examples of how to use the NIO text file methods to read and write text files. It also shows you how to use the Swing JFileChooser component which allows he user to pick a file to open. Another example shows how to read a delimited .csv file, which you will do here.

You can use these programs as templates for text file handling programs. We will do this frequently in the labs.

Part 1: Person

Create a new IntelliJ project called **Practicum01** add it to GitHub control as a public repo.

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 src folders.) The SafeInput.java source file is available with the assignment directions in Canvas. (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.

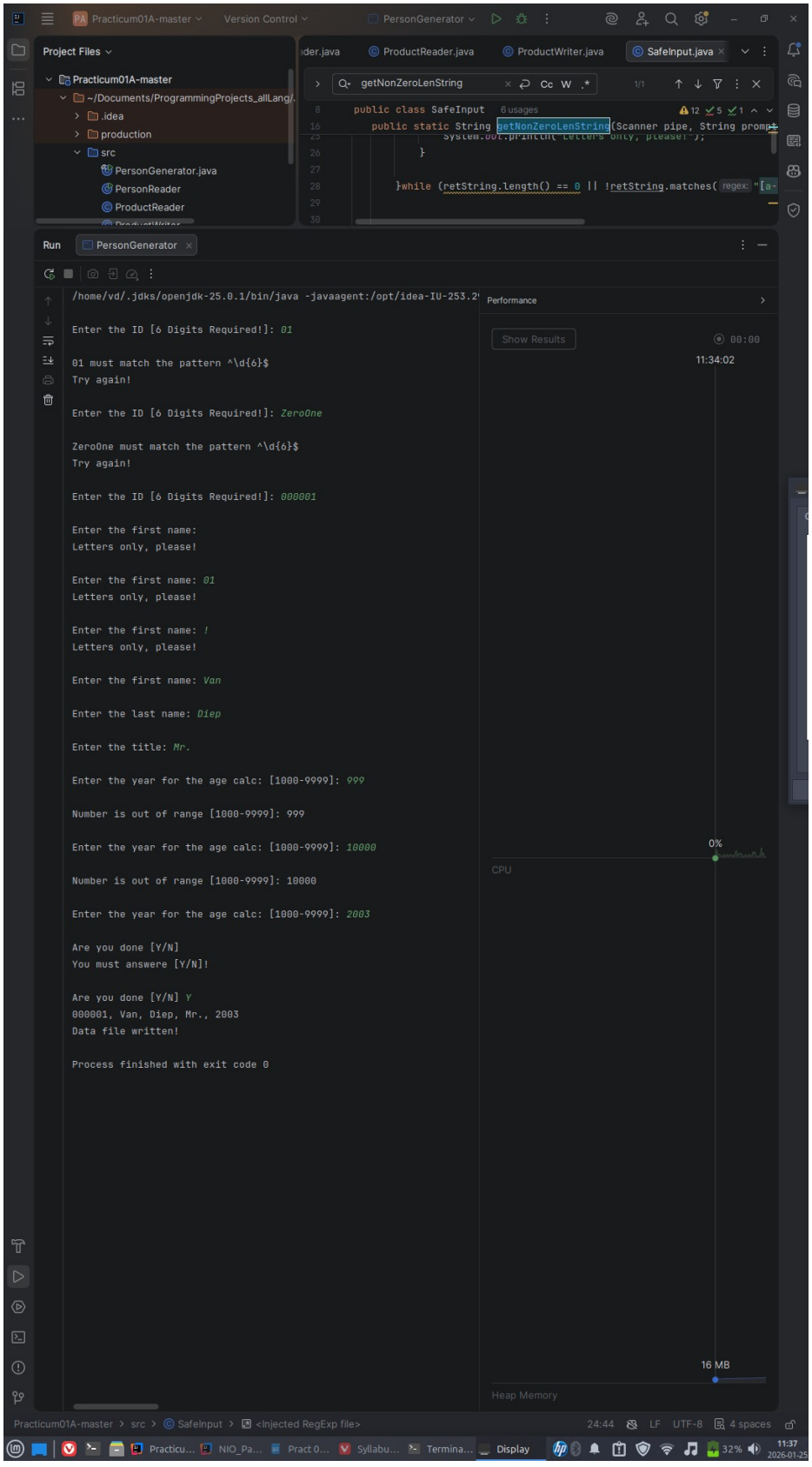

Figure 1: Possible inputs via SafeInput

5. Here is the data and file format. One record per line. Each line has all of the data elements (a – e)
    a. ID (a String)
    b. FirstName
    c. LastName
    d. Title (a string like Mr., Mrs., Ms., Dr., etc.)
    e. YearOfBirth (an int)
6. 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

7. 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

8. 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!

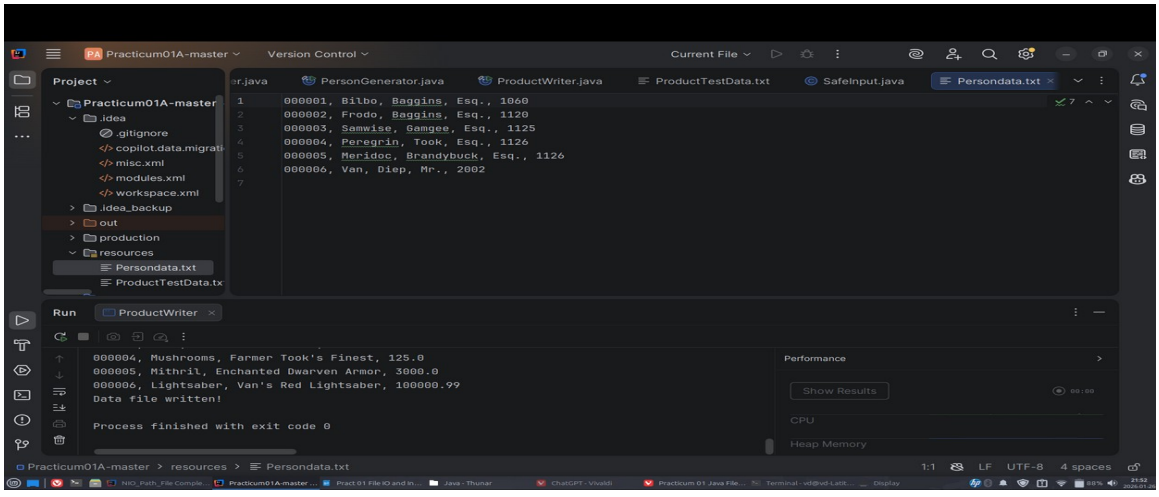

Figure 2: Persondata.txt displayed on IntelliJ Editor. Created via PersonGenerator.java

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

   000002 Frodo Baggins Esq. 1120

   …

   GET SCREENSHOTS OF:
   - THE FILECHOOSER RUNNING
   - THE DISPLAY OF THE CHOOSEN FILE

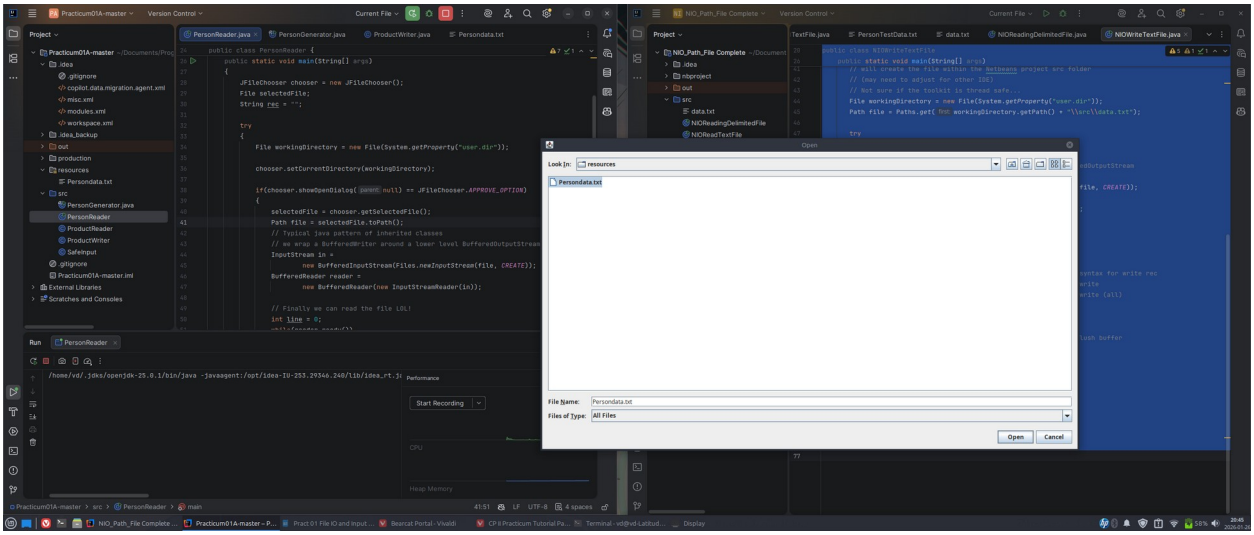

Figure 3: FileChooser Running

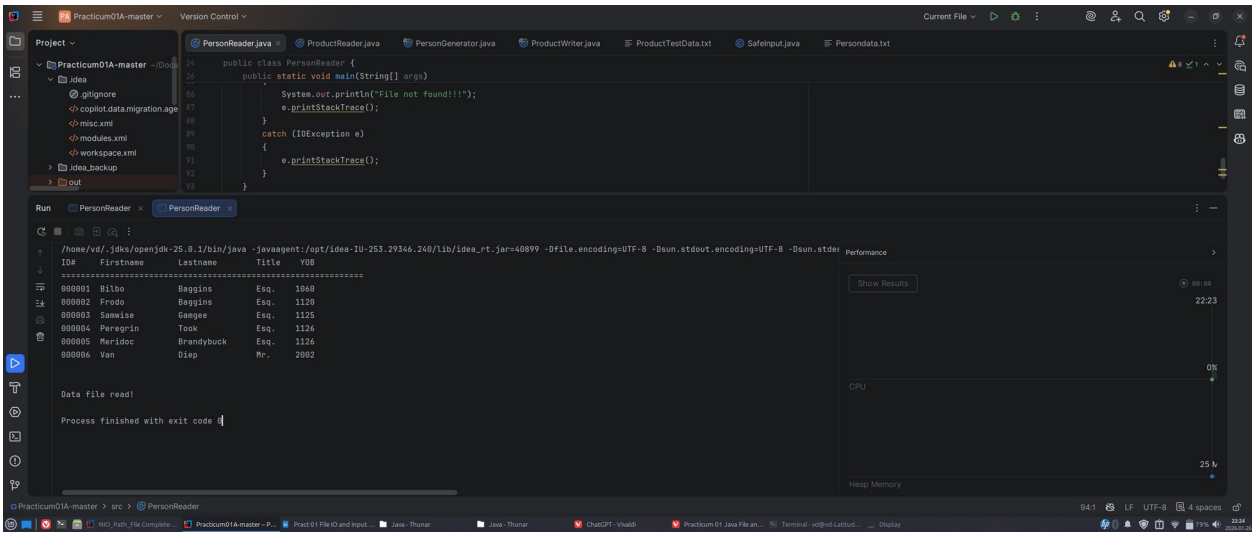

Figure 4: Persondata.txt being read and formatted after selecting it from FileChooser

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:
    a. ID (a String as before in Person)
    b. Name (a String)

c. Description (a String a short sentence)
d. 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

Part 4: ProductReader.java

4. Implement ProductReader to use a formatted display as you did in PersonReader.
5. Remove any other test files but leave a copy of this file in your project.
6. **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.**

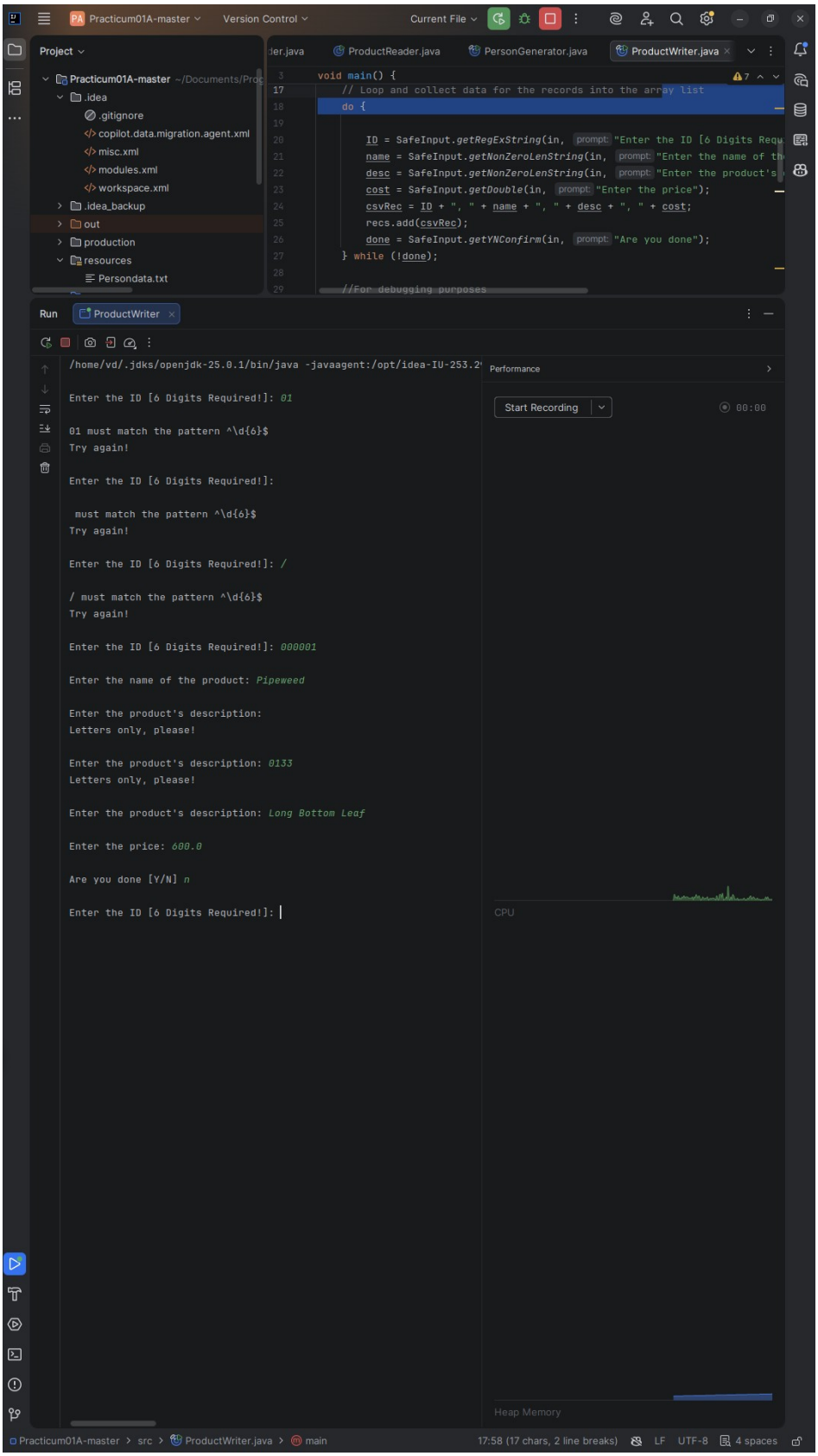

Figure 5: ProductWriter Input testing with SafeInput

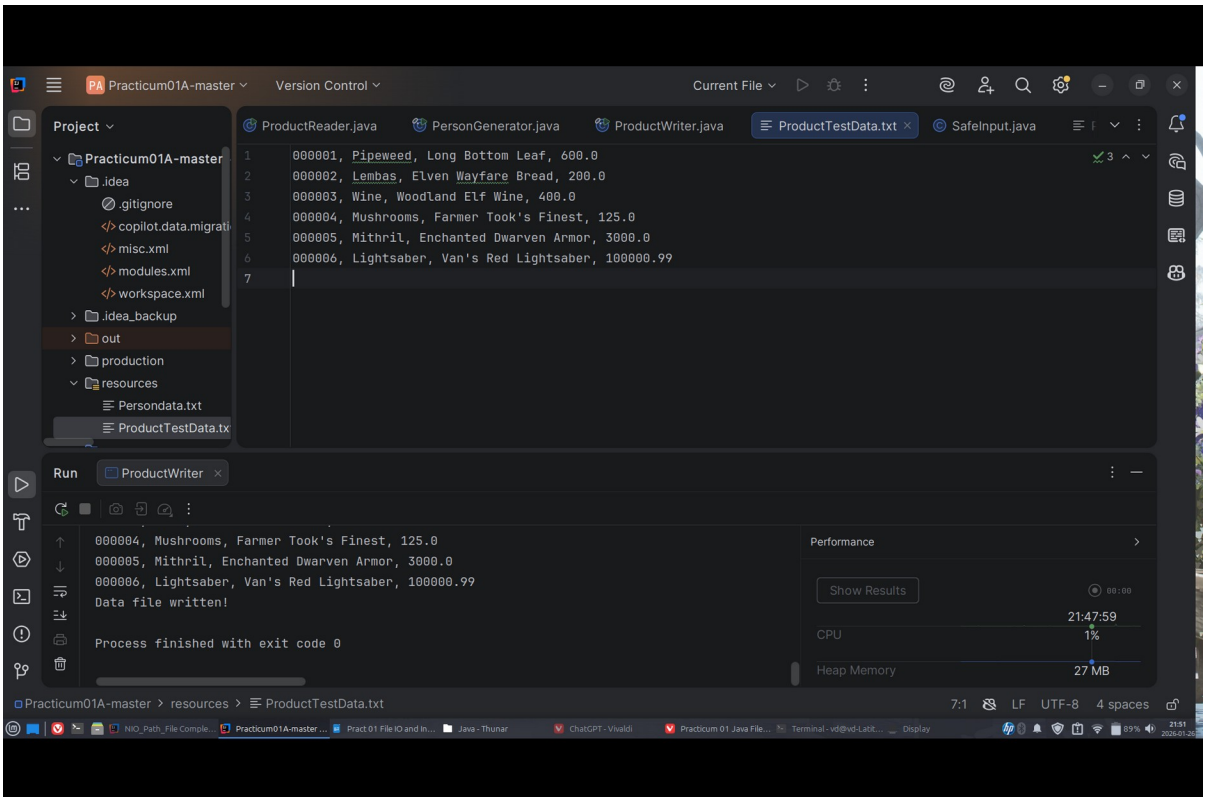

Figure 6: ProductTestData.txt displayed in IntelliJ Editor after inputting in data. Created via ProductWriter.java

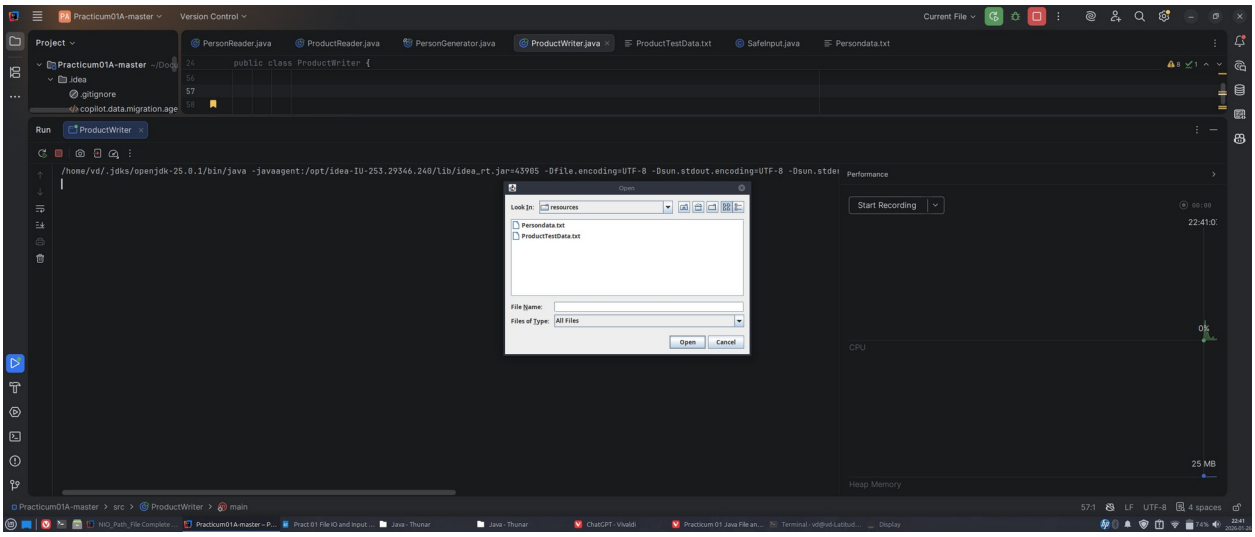

Figure 7: Filechooser being displayed for ProductWriter.java

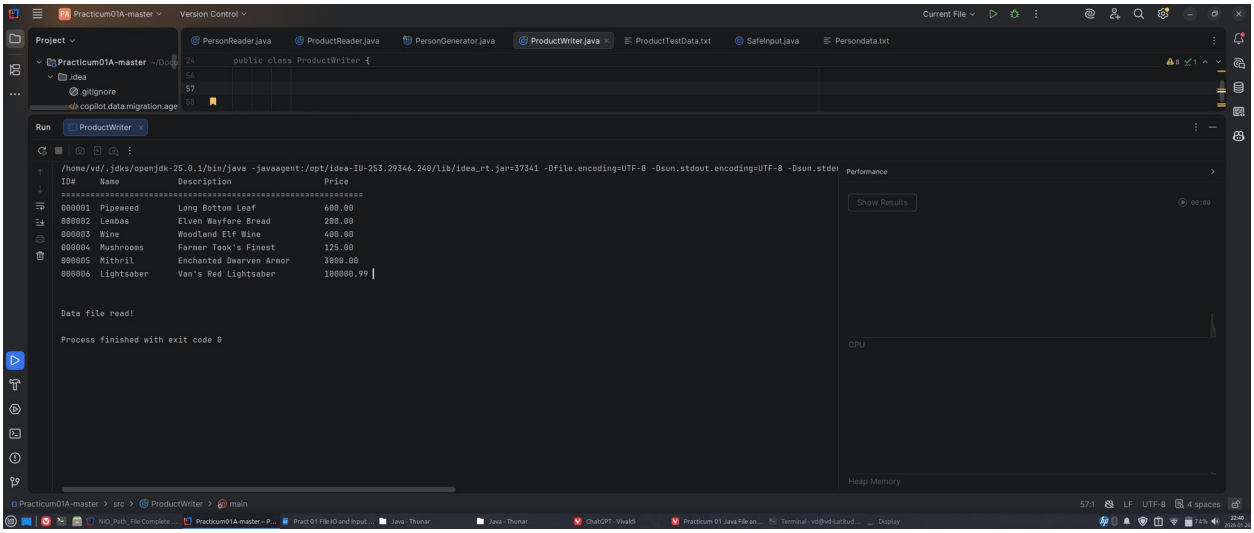

Figure 8: ProductTestData.txt being read and formatted after selecting it from FileChooser

https://github.com/diepveetee/Diep_Van_Pract01_Review.git

Submission:

Save this file as **Lastname_Firstname_Pract01_Review.docx** using your own name.

DO NOT INLCUDE SEPARATE SCREEN SHOT FILES INSTEAD OF EMBEDDING THEM HERE IN THE DOCUMENT WHERE INDICATED!

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 without permission.