Homework 8 – The ExceptIOnal Attendance Taker

Problem Description

Hello! Please make sure to read all parts of this document carefully.

In this assignment, you will create an attendance taker using files and exceptions. You will create **AttendanceTaker.java** to represent the system and create three exceptions: **BadFileException.java, InvalidNameFormatException.java, and InvalidAttendanceInformationException.java**. Let's get to work!

To complete this assignment, you will use your knowledge of exceptions and file I/O.

Remember to test for Checkstyle Errors and include Javadoc Comments!

Solution Description

Part 1: Exceptions

Create **BadFileException**, an unchecked exception, and **InvalidNameFormatException** and **InvalidAttendanceInformationException**, two checked exceptions, each in their respective file. All of these are concrete classes and will extend directly from Exception or RuntimeException (use the information on checked/unchecked to determine which exceptions extend from which class).

Part 2: AttendanceTaker

Variables

Variables must be not allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. **Hint**: there is a specific visibility modifier that can do this!

The AttendanceTaker class must have these variables.

- `File inputFile`
- `File outputFile`

Constructors

- A constructor that accepts two File objects (inputFile and outputFile, in that order)
- A constructor that accepts two Strings to represent the filenames (again, input before output) ○ Chains to the other constructor
  - Hint: This should be a single-statement constructor
- NOTE: These constructors should not attempt to perform any operations in the files, that is, the instantiation should always be successful (for example, it should success even if the input file doesn't exist).

Methods

- Please follow the instructions exactly. Do not create additional methods or instance variables

- public void takeAttendance() ○ The first part will be processing the input file, where we will extract an array of Strings representing the names we want to use to take attendance. If this operation is not successful, no further steps should be executed.
    - ▪ The input file has a specific format, storing all its information in a single line.
    - ▪ You should open the input file with a Scanner. If the Scanner throws a FileNotFoundException, it should let it propagate. Because it's a checked exception, you will have to change the method header.
    - ▪ If the file is completely empty, you should throw a BadFileException with the following message: "The input file was empty" and end here.
- You should use nextLine in the scanner to get all the information in one go.
- When that doesn't work, it will throw a NoSuchElementException. You can catch that exception and throw a BadFileException inside the catch.
    - ▪ Otherwise, continue: A valid file has to start with **|--** and has to end with with **--|** (both have 3 characters and no spaces). If any of these isn't present, or if the file has less than 6 characters (main reason: to avoid considering **|--|** or **|---|** as valid), you should throw a BadFileException with the message: "The file doesn't have correct beginning or end"
    - ▪ If none of the previous checks trigger, the file is considered valid. Remove the beginning and end (the 6 characters) and split the remaining characters with the separator **---** (3 hyphenes, no spaces, use the split method). The resulting array of Strings will be the array having the names. With the names obtained, you should close the input file.
    - ▪ If a BadFileException or FileNotFoundException occurs, you shouldn't catch it, let it propagate. You must make sure that the input file is closed regardless of whether an exception is thrown or not.
- Hint: There's a keyword that can be used in combination with a try/catch to achieve this behavior.
    - ○ Make sure to store the array of String outside of the previous block of code so it can be accessed later. If it's a local variable inside a try block, it won't be able to be accessed after it ends. Declare the array outside the try and set it to null initially.
    - ○ For the second part, we will use a helper method to take the attendance. ○ Create a Scanner from System.in and a PrintWriter for the output file. You won't use them on this function, instead, you will pass them to a helper function that will be described after this one.

- For each name, call the helper method described below (takes the name, the Scanner for the console input, and the PrintWriter). You should catch each of the two possible exceptions it can throw and print to console a message if that happens (do nothing if the method call executes successfully) and then continue to the next name.
  - If the call results in an InvalidNameFormatException, print to console the following: "Skipping [name] because of an invalid name format: [exception message]" (no brackets)
  - If the call results in an InvalidAttendanceInformationException, print to console the following: "Skipping [name] because of an invalid attendance information: [exception message]" (no brackets)
    - For each of them, the exception message is the Exception's getMessage() value. The name is the current name being processed.
- Because you will catch all the exceptions we will test (there could be unchecked exceptions that we won't test), your try/catch (with double catch!) should be inside your loop.
- After your loop completes, close the scanner for the console input and close the PrintWriter. That's the end of this function.
- **This can be a lot to digest in a single take so don't worry if it fells too much (it would never be this complex in an exam). The steps are written (mostly) sequentially, so go step by step, and take a break if you need to! If you need help, we are available on Piazza.**
  - **Example output is provided for different scenarios to guide you in case you find the instructions unclear. Of course, you can also ask in Piazza if that's not enough!**

- private static void processStudentAttendance(String name, Scanner consoleScanner, PrintWriter printWriter) ○ NOTE: For all exceptions, before throwing it, put a **-** (hyphen) in the output file in its own line with the PrintWriter's newline – NOT THE System.out!!! - before throwing the exception (this will happen in 4 scenarios, so this note is to avoid the 4x repetition)
  - The first thing we will do is a bunch of checks in the name. If any of these fails, an exception will be thrown and propagated.
    - If the name isn't "uppercase only" (String must match with its toUpperCase()), throw an InvalidNameFormatException with the message "The name isn't uppercase only"
    - Otherwise, we character by character, and perform these checks in the following order:
- If the character is a digit, throw an InvalidNameFormatException with the message "The name has a digit"
- If the character is a pipe character (the **|**), throw an InvalidNameFormatException with the message "The name has a pipe character"

- If those checks pass, print “[name]: “ to console and await for user input. Use nextLine.
  - If the returned string is empty (the user just hit “enter” immediately), throw an InvalidAttendanceInformationException with the message “Attendance information missing”
  - If the returned string is not an **A** (for absent) or **P** (for present), throw an InvalidAttendanceInformationException with the message “Attendance information is not P or A”
  - Otherwise, pass the obtained string directly to the PrintWriter. Use it’s println method to print a line with just the letter to the file.

- public static void main(String[] args)
  - Create an AttendanceTaker using the first two console arguments (guaranteed to exist, no need for checks) and call its takeAttendance method.

Example 1:

File myInput.txt: **|-Ignacio--|**

> **java AttendanceTaker notMyInput.txt myOutput.txt**

Result: Your program should terminate abruptly will a FileNotFoundException thrown from the takeAttendance (and not caught in it or main). The exception should have the message “myOutput.txt should NEVER be created, a Scanner for System.in should NEVER be created, and a PrintWriter for myOutput.txt should NEVER be created. The Scanner for notMyInput.txt should be closed.

Example 2:

File myInput.txt: **|-Ignacio--|**

> **java AttendanceTaker myInput.txt myOutput.txt**

Result: Your program should terminate abruptly will a BadFileException thrown from the Scanner’s nextLine in takeAttendance (and not caught in either takeAttendance or main). The exception’s message should be “The file doesn't have correct beginning or end” as the 3rd character from the beginning is missing. myOutput.txt should NEVER be created, a Scanner for System.in should NEVER be created, and a PrintWriter for myOutput.txt should NEVER be created. The Scanner for myInput.txt should be closed.

Example 3:

File myInput.txt: **(an empty file) <- NOT text: a file with no bytes.**

> **java AttendanceTaker myInput.txt myOutput.txt**

Result: Your program should terminate abruptly will a BadFileException thrown from the Scanner’s nextLine in takeAttendance (and not caught in either takeAttendance or main). The exception’s message should be “The input file was empty” as the file is empty. myOutput.txt should NEVER be created, a Scanner for System.in should NEVER be created, and a PrintWriter for myOutput.txt should NEVER be created. The Scanner for myInput.txt should be closed.

Example 4:

File myInput.txt: **|--NAME---NAME2---name---MYNAME---NA|ME---MYNAMEAGAIN--|**

> **java AttendanceTaker myInput.txt myOutput.txt**

Console I/O. User input in bold

NAME: **P**

Skipping NAME2 because of an invalid name format: The name has a digit

Skipping name because of an invalid name format: The name isn't uppercase only

MYNAME: **A**

Skipping NA|ME because of an invalid name format: The name has a pipe character

MYNAMEAGAIN: **[USER HITS ENTER WITHOUT A CHARACTER]**

Skipping MYNAMEAGAIN because of an invalid attendance information: Attendance information missing

Expected output file: myOutput.txt

P

-

-

A

-

-

Result: Your program should terminate successfully with the shown console I/O and correct output in myOutput.txt. The file myOutput.txt will be created or overwriten (it will have 6 lines or 7 with the last line being an empty line). Both Scanners and the PrintWriter should be closed.

Rubric

See autograder tests. All tests are public.

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import the following classes. You are *not* allowed to import any other classes or packages.

- `java.io.File`
- `java.io.FileNotFoundException`
- `java.io.PrintWriter`
- `java.util.NoSuchElementException`
- `java.util.Scanner`
- `While we won't use and you shouldn't import IOException, it's an important exception in this course. Remember it!`

Feature Restriction

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Javadoc

For this assignment, you will be commenting your code with Javadoc. Javadoc comments are a clean and useful way to document your code's functionality. For more information on what Javadoc comments are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the Javadoc overview for your code using the command below, which will put all the files into a folder named "javadoc". Note you should execute this after adding Javadoc comments to your code:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;


/**

 *This class represents a Dog object.

 *@author George P.Burdell
 *@version 1.0
 */  public class
Dog {     /**

     *Creates an awesome dog(NOT a dawg!)
     */      public
Dog() {
     ...
     }


     /**
      *This method takes in two ints and returns their sum
      *@param a first number
      *@param b second number
*@return sum of a and b
      */       public int add(int
a,int b) {
      ...
      }
}
```

A more thorough tutorial for Javadoc Comments can be found here.

Take note of a few things:

1. Javadoc comments begin with /** and end with */.
2. Every class you write must be Javadoc'd and the @author and @verion tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an @param tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadoc comments using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here.

To run Checkstyle, put the jar file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **30 points**.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials. or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradecope:

- AttendanceTaker.java

- BadFileException.java
- InvalidNameFormatException.java
- InvalidAttendanceInformationException.java

Make sure you see the message stating "Homework 8 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

All test cases will be public tests, like HW2 and HW6.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications