Regis University CC&IS
CS210 Introduction to Programming
Java Programming Assignment 7: Files and Exceptions

NOTE: Be sure to follow the testing requirements, so you don't lose points!

Problem Summary

An investment firm is interested in having you modify the last program you wrote for calculating compounded interest earnings. They would like you to change the program to process a data file, and determine how much interest will be paid to their customers' accounts annually.

Each line of the data file will contain a 5-digit account number, a beginning balance, an annual interest rate, and a compounding type. Sample data file lines would be:

```
11111 1111.11 11.1 A
22222 222.22 2 Q
33333 3333.33 13 D
```

You will write a program to:

- Try to open a user-specified text input file and a specific text output file, and check for errors.
- If there are no errors opening the files:
  - Read data from the input file. For each line:
    - Error check each line for valid data, and ignore lines that contain invalid data.
    - For valid data lines, calculate interest earned and write the results to an output file.
  - Display a summary of the number of valid accounts and total interest paid to the screen.

Since data will no longer be entered by the user, the program will implement exception handling to error check the values from the input data file are valid.

Program Requirements

Required Classes and Methods

Two separate classes will be required for this program.

1. The first class will be the same **Investment** class you defined for Java Assn 6.

   Within the class:

   - The class will still have the same private data properties, with a couple of changes:
     - The **initial investment** data property will be modified to be called a **beginning balance**.
     - The **balance** data property will be modified to be called an **ending balance**.
     - A ***new*** data property, for the **account number**, will be added.
   - Add a **getter** for the **account number**.
   - Modify the ***second*** constructor (the one that has parameters for ***four*** of the data properties):
     - Remove the parameter for the **term**.
     - Add a new parameter for the **account number**.
     - Move the constants that define the minimums and maximums for the beginning balance and annual interest rate data fields from the **main** class to this constructor.

- Implement Exception Handling for ***each*** of ***three*** parameters, as follows:
  - If the **beginning balance** parameter is invalid (not in range of $1 to $500,000)
    - Throw an **IllegalArgumentException**
    - Include a message to be passed back to the exception handler that explains why the starting balance is invalid, explains that the data from this line will be ignored, and ***includes all of the data items from the bad data line***.
  - If the **annual interest rate** parameter is invalid (not in range of 1% to 30%)
    - Throw an **IllegalArgumentException**
    - Include a message to be passed back to the exception handler that explains why the interest rate is invalid, explains that the data from this line will be ignored, and ***includes all of the data items from the bad data line***.
  - If the **compounding type** parameter is invalid (i.e. not 'A', 'Q', 'M' or 'D')
    - Throw an **IllegalArgumentException**
    - Include a message to be passed back to the exception handler that explains why the compounding type is invalid, explains that the data from this line will be ignored, and ***includes all of the data items from the bad data line***.

  NOTE: All of these exceptions will be **caught** and processed by the method that called the constructor.
- If there are no exceptions thrown, the constructor will:
  - Set the account number, beginning balance, annual interest rate, and compounding type data fields directly, using the passed in parameter values.
  - Set the **term** data field to 1 (new).
  - Set the **ending balance** data field to the same value as the **beginning balance** (modified).
  - Set the **earnings** data field to 0 (same as before).

2. The second class will be the main class that contains the **main** method.

   NOTE: This class will be ***different*** from Java Assn 6's main class, so you will need to delete the old **InvestmentComparison** class file.

   Create a ***new*** main class to process the data file, write results to a file, and display a summary. Name the new class:

   `InterestPaidCalculator`

   Within the **InterestPaidCalculator** class:

- Create a **main** method that will:
  - Define the following constants and local variables:
    - A constant containing the output data file name **`report.txt`**
      - Do not supply any pathnames with this file (just use **report.txt**).
    - Variables to hold input **File** and **Scanner** objects, and initialize them both to **null**.
    - Variables to hold output **File** and **PrintWriter** objects, and initialize them both to **null**.
    - A variable to hold the **String** input text data file name.
    - Two **boolean** variables to indicate if you can open the input data file and the output data file.

- A variable for a **Scanner** object, instantiated to read from the keyboard.
- Variables to hold the number of accounts processed and the total interest paid by the investment company in one year, both initialized to 0.

- Display a description to the user, describing what the program will do.
- Read the name of the input text data file from the user.
- Try to open the input text data file
  - If the file opened, set the boolean variable indicating the open was successful.
  - Catch any **FileNotFoundException** exceptions and display a message that includes the name of the file and that it could not be opened.
- Try to open the output text data file
  - If the file opened, set the boolean variable indicating the open was successful.
  - Catch any **IOException** exceptions and display a message that includes the name of the file and that it could not be created.
- If both files opened (no exceptions):
  - Display a message to the user, that the program is reading the file and calculating the interest paid.
  - In a loop, until you reach the end of the file:
    - Read the four data items from one line of data in the text file.
      NOTE: You can assume that each line will always contain one integer, two doubles, and one character.
    - Try:
      - ✓ Create a new **Investment** object, using the data read from the file as the input parameters to the constructor.
      - ✓ If no exceptions are thrown (within the **try** clause):
        - Using the object, call the instance method to calculate the investment earnings (from Assn 6).
        - Using the object and getters, write the account number, and earnings to the report file. The earnings should be rounded to 2 decimal places, and all decimals should line up.

          Example (for the first line of the sample data file):

          ```
          11111        123.33
          ```

        - Add the earnings total to the total interest paid by the investment company in one year
        - Increment the number of accounts processed.
    - Catch any **IllegalArgumentException** exceptions and display the message passed back. Then continue reading the next line of data.
  - Close the files.
  - Display the number of accounts processed (include valid lines read only).

        - Display the total interest paid by the investment company for the year.
    - If either file could not be opened, just exit.

WARNING: The objects, classes, and methods must be implemented exactly as specified above. If your program produces correct output, but you did not create and use the objects as specified, and implement the required classes and methods, you will lose a significant number of points.

Sample Run 1 (using the 3 lines of sample data shown on first page):

```
This program calculates the interest paid by an investment
company, from all accounts, over one year.

Enter input data filename (including .txt):
accounts1.txt

Reading file and calculating interest....

Results:
    For 3 investment accounts, the investment company paid
    $ 590.48 total interest.
```

Sample file with some data line values that are not valid:

```
55555 55 5 Q
12345 0.0 3.5 A
88888 888 8.8 D
54321 1000.00 10 X
```

Sample run 2 (using file above with invalid data):

```
This program calculates the interest paid by an investment
company, from all accounts, over one year.

Enter input data filename (including .txt):
accounts2.txt

Reading file and calculating interest....

  ***Invalid beginning balance 0.0 -- data line:  0.0 3.5 A ignored
  ***Invalid compounding type X -- data line:  1000.0 10.0 X ignored

Results:
    For 2 investment accounts, the investment company paid
    $ 84.48 total interest.
```

Coding Standards

The program must follow the **CS210 Coding Standards** from Content section 6.10.

Be sure to ***include*** the following comments:

- Comments at the ***top of each code file*** describing what the class does

        - Include **tags** with the author's name (i.e. your full name) and the version of the code (e.g. version 1.0, Java Assn 7)
    - Comments at the ***top of each method***, describing what the method does
        - Include **tags** with names and descriptions of ***each*** parameter and return value.

Delete any default comments supplied by the IDE that you did not use.

Testing

You will need to create test data files to test your program. Your test data files should test every possible execution path within your code, including erroneous data which cause exceptions.

For example, you could name the file **`accounts.txt`**. It will need to be placed in the top level of your **project** directory, in order for your program to find it.

> WARNING: Test data files that simply mimic the test data files from this requirements document will NOT be counted. You must come up with **your own original** test data.

Before you submit your project, add your all files you used to test your program in the top level of your project directory (and add a number to the end of each file, if there are multiple test data files).

File examples: **`accounts1.txt`**
`accounts2.txt` (numbered, if multiple data files tested)

Submission

This programming assignment is due by midnight of the date listed in the **Course Assignments by Week**.

> Programs submitted that **do not compile** without errors **will not be accepted**.

Again, you will submit a single **zip** file containing all of the files in your project.

- First export your project from NetBeans, as detailed in previous assignments
    - Name your export file in the following format:
      `<lastname>Assn<x>.zip`

      For example: **`SmithAssn7.zip`**

      NOTE: Save this zip file to some other directory, not your project directory.
- Then submit your **.zip** file to the **Java Prog Assn 7** assignment submission folder (located under the **Assignments/Dropbox** tab in the online course).

    - Warning: Only NetBeans export files will be accepted.
      Do not use any other kind of archive or zip utility.

Grading

Programs will be graded using the **rubric** that is linked on the same assignment page as this file.

WARNING:

*Programs submitted more than 5 days past the due date will **not** be accepted, and will receive a grade of 0.*