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

Problem Summary

The credit card company that you wrote the last program for would like you to expand the program to process a data file. They would like to have a program that will help them determine how much money they will earn in interest from their customers. The program will now handle all starting balances (even if they are under $500).

Each line of the data file will contain a starting balance, an annual interest rate, and a percent of the balance to pay off each month.

Sample data file lines would be:

```
1111.11 11.1 11
222.22 2 22
3333.33 13 33
```

You will write a program to:

- Read a number of months to calculate for, from the credit card company.
- Read data from a file, and determine the total interest paid by each customer for those months.
- Provide a report on the credit card company profits.

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.

NOTE: For an example of a Java program using file data, see Online Content section 13.11.

Program Requirements

Required Classes and Methods

Two separate classes will be required for this program.

1. The first class will be the same Credit Card Account class you defined for Java Assn 6.

   For example: `CreditCardAccount`

   The class will still have the following **private** properties (no changes):

   - current balance
   - annual interest rate (percentage – e.g. 12.5%)
   - percent of current balance to pay each month (e.g. 10%)

   Within the class:

   - This program will use only the ***second*** constructor that has parameters for ***all three*** of the properties. Modify that constructor, as follows:
     - Implement Exception Handling for ***all*** of the parameters
       - If the starting balance read is invalid (i.e. not above $0)
         - 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 interest rate read is invalid (i.e. not between 3% and 25%)
  - 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 pay down percent read is invalid (i.e. not between 0% and 33%)
  - Throw an **IllegalArgumentException**
  - Include a message to be passed back to the exception handler that explains why the paydown percent is invalid, explains that the data from this line will be ignored, and ***includes all of the data items from the bad data line***.
- Otherwise, the constructor will initialize all of the data fields to the parameter values.

NOTE: All of these exceptions will be caught and processed by the method which calls the constructor.

- Modify the instance method to make a payment. This method will now:
  - Record a payment for one month, doing all the same calculations as before, but ***without*** displaying anything.
  - Return the amount of interest paid for the month.
- Modify the payoff instance method.

  First rename the method to **calcAccountInterest**. This method will now calculate the interest paid on an account, over a specific number of months.

  Then modify the method code to:
  - Have one parameter, a number of months to calculate interest for.
  - Remove all code that displays anything.
  - Change the loop to a **for** loop to calculate payment information each month, as follows:
    - Call the method to make a payment
    - Add the returned interest for the month to the total for the account.

    Stop looping after making payments for the number of months specified by the parameter OR when the account balance reaches 0.
  - Return the total interest paid on the credit card account for all the months specified.

2. The second class will be the main class that contains the **main** method.
   This class will be ***different*** from Java Assn 6's main class, so delete the **`PayoffComparison`** class.

   This new main class will process the data file and display the results. Name it:

   `ProfitCalculator`

   Within the **ProfitCalculator** class:

   - Define a **static** method to prompt for, read, and validate the number of months to calculate interest for, as follows:
     - Prompt for a number of months greater than 0.

- Error check, issuing error messages as necessary, and loop until a valid value is entered.
- Return a valid number of months.

- Create a **main** method to:
  - Define a local constant containing the input data file name of **`accounts.txt`**
    - Do not supply any pathnames with this file (just use **`accounts.txt`**).
  - Display a description to the user, describing what the program will do.
  - In a loop that will repeat the rest of the method code:
    - Call the static method to read the number of months to calculate the interest for
    - Try to open the input text data file
      - Catch any **FileNotFoundException** exceptions and display a message that includes the name of the file and that it could not be opened.
        Then exit the program.
    - If the file opened (no exceptions):
      - In a loop, until you reach the end of the file:
        - ✓ Read three data items from one line of data in the text file.
        - ✓ Try to create a new Credit Card Account object, using the data read from the file as the input parameters to the constructor.
          - The **try** clause should include all the code that uses this object.
          - Catch any **IllegalArgumentException** exceptions and display the message passed back. Then continue reading the next line of data.
        - ✓ Call the **calcAccountInterest *instance*** method with the object and add the returned interest for the account to the total interest earned by the credit card company.
      - Close file.
      - Display the number of accounts processed (valid lines read).
      - Display the total interest earned by the company within the specified number of months
  - Ask the user whether to repeat the program, using a different number of months for the interest calculations.
  - Repeat the program until the user says they do not want to.

Sample Run (using the 3 lines of sample data shown above):

```
This program calculates how the interest earned by a credit card
company, from all accounts, over a specific number of months.


Enter the number of months to calculate interest for: 6

  ***Invalid card interest 2.0% -- data line:  222.22 2.0 22.0 ignored

Over the next 6 months,
     the total interest earned from 2 accounts will be $ 149.04

Run again with a different number of months (y/n)? y

Enter the number of months to calculate interest for: 12

  ***Invalid card interest 2.0% -- data line:  222.22 2.0 22.0 ignored

Over the next 12 months,
     the total interest earned from 2 accounts will be $ 183.17

Run again with a different number of months (y/n)? n
```

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.

3. 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. Java Assn 4, version 1.0)
   - Comments at the ***top of each method***, describing what the method does
     - Include **tags** with names and descriptions of ***each*** parameter and return value.

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. Remember the data file should be named **`accounts.txt`**, and will need to be placed in the top level of your **project** directory, in order for your program to find it.

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:
  - Highlight the project name.
  - Click on **File** from the top menu, and select **Export Project**.
  - Select **To ZIP**
  - 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.*