Regis University CC&IS
CS210 Introduction to Programming
Java Programming Assignment 8: Files, Exceptions, and Arrays

NOTES:
Again, this assignment will be ***more challenging*** than the previous assignments, so make sure to ***start early*** and ***allocate enough time*** to complete it, including time you might need to seek any necessary help.

And be sure to follow the testing requirements, so you don't lose points!

This last programming assignment may not be submitted more than **2 days** late

Problem Summary

A cell phone company specializes in cell phones for single people (i.e. plans for a single line). They would like to hire you to write a program to help them calculate cell phone bills and analyze data.

Each cell phone bill includes three components:
Phone charges (as defined below) for each cell phone
State and local taxes of 8.1% tax on phone charges (Note: 8.1% would be stored as 0.081)
$ 1.44 Universal Service fee

Phone charges are based on the type of plan. Two plans are available, and both include unlimited talk minutes.
P = Pay as you go plan (no texts or data included)
U = Unlimited talk & text plan (no charge for texts, 2 GB of data included)

Phone charges are as follows for each plan:

Plan	Base Fee	Charge for texts	Charge for data
P	5.99	0.10 per text	$3.39 per GB*
U	59.99	No charge	$2.29 per GB over 2 GB*

*Charged by whole GB (not partial GB)
So be sure to round up to the nearest whole value before calculating charges

The company collets cell phone usage data monthly, and stores the data for each phone in a text data file. For each cell phone account, the file holds:
- The cell phone number – a 12-character String (###-###-####)
- The plan type (uppercased letter)
- The number of text messages sent and received
- The number of GB of data used, to one decimal place (e.g. 1.5)

The number of lines in the data file is unknown, but will be less than 500.

Sample input data file lines:
```
303-333-3333 P 50 0.9
720-777-7777 U 444 1.5
720-222-2222 P 200 1.8
720-123-4567 U 2000 4.4
```

You may assume all data in input data file is valid data (i.e. formatted correctly), but you must verify that the file exists (i.e. can be opened). If the file does not exist, the program will loop until the user enters the filename for a file that can be opened.

The company would like you to write a program to read the file, calculate the account's monthly bill, and generate several reports.

The program will:

- Read the month's data from the file, line by line, and store the data into an array that holds cell phone account data objects (maximum of 500 cell phone account data objects).
- Calculate the month's total bill, based on the cell phone account data.
- Produce an output report file, containing the phone numbers and month's total bill for each account, along with the total collected from customers.
- Analyze the bills to determine the high, low, and average charged, and display the results.

Program Requirements

Overview of Required Classes and Methods

Three separate classes will be required for this program.

- A class to define a **CellPhoneAccount**
  - Contains data fields, constructor, setters, and getters
  - Defines method to calculate the bill total for one **CellPhoneAccount** object

- An **AccountsListImpl** class to implement the account array (to hold **CellPhoneAccount** objects)
  - Contains data fields for the array and the number of objects in array
  - Contains constructor and getter
  - Defines method to add a **CellPhoneAccount** object to the array
  - Defines method to read file data and use it to create objects and put them in the array
  - Defines method to calculate bill totals for all **CellPhoneAccount** objects in the array

- A **BillingAnalysis** ***main*** class to run the program
  - Instantiates an **AccountsListImpl** object
  - Reads input data filename and uses **AccountsListImpl** object to call method to read the file data and fill the array.
  - Uses **AccountsListImpl** object to calls methods to calculate all bill totals and write a billing report to a file.
  - Calls static method to display a billing analysis summary
  - Loops until user wants to quit

Project Requirements

Create a NetBeans **project** named: **`LastnameAssn8`**
with a **main** class named: **`BillingAnalysis`**

Then define the classes and methods as follows:

1. Define a Java **class** with properties and methods for a cell phone account object, named: **`CellPhoneAccount`**

   The class will have a **private** data property for each item on the text file data line.
   (i.e. Cell phone number, plan type, number of text messages sent and received, and number of GB of data used)

   Additionally, it will have a **billTotal** private data property to hold the month's bill total.

Within the **CellPhoneAccount** class:

- Define a **constructor**, with parameters for each data property that is read from the text data file.
    - Use the parameters to initialize the values.
    - Initialize the **billTotal** to 0.
- Define **getters** and **setters** for each data property.
- Define an **instance** method to calculate and store the **billTotal** for a **CellPhoneAccount** object. The method will:
    - Define constants for the calculations (from page 1 of this assignment).
        - Constants will be defined for **all** fixed values (including **base rates, included amounts, overage charges, universal fee,** and **tax rate**). Be sure that you do not embed any constant values in the constant names.
    - Use the data field values to calculate the month's total phone bill.
    - Store the calculated value into the **billTotal** data property for the object.

2. Define a second class that will ***implement the cell phone accounts array*** for this program, named:

    `AccountsListImpl`

    The class will have the following **public** data properties:

    ❖ A **static** constant array size (maximum items that can be held in the array) set to 500

    The class will have the following **private** data properties:

    ❖ An **accountArray** array (to hold **CellPhoneAccount** objects)

    ❖ A property to hold the number of CellPhoneAccount objects stored in the array

    Note: The data property definitions will only ***define*** the array reference variable.
    It will not create or initialize the array object.

Within the **AccountsListImpl** class:

- Define a **constructor** that will:
    - Instantiate an **accountArray** array object (using **new** to initialize the array reference variable and the constant array size)
    - Initialize the number of **CellPhoneAccount** objects stored in the array to 0
- Define a **getter** to return the number of objects stored in the array
- Define an **instance** method to **add** one **CellPhoneAccount** object to the **accountArray**. The method will:
    - Have one parameter, the **CellPhoneAccount** object to add
    - Test to see if the array is full
        - If the array is full, will throw an **ArrayIndexOutOfBoundsException**
            - Include a message to pass to the exception handler that states the array is full and stating which phone number cannot be added.
        - Otherwise, store the **CellPhoneAccount** object into the **accountArray** and increment the number of objects stored in the array.

- Define an **instance** method to read and store all the data from the input file. The method will:
    - Have one parameter: the name of the data input file (String)
    - **Try** to open the input data file.

      If the file opened:
        - Display a message that the program is reading and storing the file data
        - In a loop:
            - Read all data items from one line of the input data file
            - Call the constructor to instantiate a new **CellPhoneAccount** object, passing in the data read from the file.
            - **Try** to call the instance method to add the **CellPhoneAccount** object to the array (note: this is an instance method of the **AccountsListImpl** class)
            - **Catch** any thrown **ArrayIndexOutOfBoundsException** exceptions.
                - Display the message passed back via the throw.
                - Also display a message that no more data will be read from the file.
                - Set a value to exit the loop (so no more data will be read).

          Loop until there is no more file data or an exception is thrown.
        - Close file.
        - Display the number of **CellPhoneAccount** objects stored in the **accountArray**.
    - **Catch** any **FileNotFoundException** exceptions. When caught:
        - Display a message that says which file could not be opened.
- Define an **instance** method to loop through all the objects in the **accountArray** and calculate and store values for the **billCharge** data field. The method will:
    - For each **CellPhoneAccount** object stored in the array:
        - Call the instance method to calculate the month's total phone bill (this is an instance method within the **CellPhoneAccount** class)
- Define an **instance** method to produce a billing report of monthly bill totals.

  The method will create an **output file** with the billing report. The method should:
    - Read the filename for the report output file from the user.
    - **Try** to open the report output file
        - If the file opened successfully, generate a report and write it to the output file.
        - The output file report will contain a list of cell phone numbers and the bill charges associated with that phone numbers (accessed via **getters**), with the charges right-aligned on the decimal point.
        - The last line of the output file will contain the total of all phone bill charges collected, right-aligned on the decimal point with the charges listed above it.

          (see sample file output on next page)

Sample output file (using data from sample input data lines above)

```
303-333-3333      16.98
720-777-7777      66.29
720-222-2222      36.86
720-123-4567      73.72
Total            193.85
```

- **Catch** any **FileIOException** exceptions. When caught:
    - Display a message that explains which file could not be opened and that a bill report will not be generated.

- Define three additional **instance** methods to:
    - Determine and return the lowest bill charge in the **accountArray**
    - Calculate and return the average bill charge in the **accountArray**
    - Determine and return the highest bill charge in the **accountArray**

  Note: You will need to use a **getter** to access the bill total in each object.within the array.

3. Within the **BillingAnalysis** main class:

- Define a static method to display a billing analysis summary, using the previously defined instance methods to calculate the low, high, and average bill charges.
    - Parameters: the **AccountsListImpl** object
    - The billing summary will be displayed as shown in the sample output on the next page.
        - All figures will be right-aligned on the decimal point.

- Define a **main** method to:
    - Display a description of what the program will do to the user.
    - In a loop (outer loop):
        - Create a new object of the **AccountsListImpl** class.
        - In a loop (inner loop):
            - ❖ Read the filename for the input data file from the user.
            - ❖ Using the object, call the instance method to read and store the data from the input file (sends a message to the **AccountsListImpl** object).
            - ❖ Get the number of **CellPhoneAccount** objects stored in the array. Note that if the file could not be opened successfully, the data field that holds the number of **CellPhoneAccount** objects stored in the array will remain 0.

          Loop, trying to read files, until the number of **CellPhoneAccount** objects stored in the array is ***not*** 0, indicating the file was read (i.e. the array now has data stored in it).
        - Display a message to the user, that the program will calculate the monthly bills.
        - Use the **AccountsListImpl** object to call:
            - ❖ The instance method to loop through all the objects in the **accountArray** and calculate and store values the **billCharge**.
            - ❖ The instance method to create the billing report of monthly bill totals.

- Display a blank line or two.
- Call the static method to display a billing analysis summary.
- Ask the user whether to run the program again, using a different input file.

Loop until the user says s/he does not want to run the program again.

Sample Run (using sample input data lines from above)

```
Program will calculate cell phone bills, produce a report of bill totals,
and display a billing analysis summary.

Enter input data filename: x
    Cannot open input file x
Enter input data filename: data.txt

Reading cell phone account data...
Data stored for 4 cell phone accounts.

Calculating all phone bills...
Enter name of report output file: report.txt

Monthly Billing Analysis for 4 cell phone accounts:
    Lowest bill charge           16.98
    Average bill charge          48.46
    Highest bill charge          73.72

Run again (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.

See last pages for outline of code for the three required classes.

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 4)
- 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.

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 files.

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: **`SmithAssn8.zip`**

  NOTE: Save this zip file to some other directory, not your project directory.

- Then submit your **.zip** file to the **Java Prog Assn 8** 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 2 days** past the due date will **not** be accepted,*
and will receive a grade of 0.

```
/**
 * Description of class and author and version tags
 */
public class CellPhoneAccount {
      // data fields go here

      // constructor with parameters (and comments) goes here

      // getters and setters (w/comments and parameter/return tags) go here

      // method to calculate and store a bill total goes here
}
```

```
/**
 * Description of class and author and version tags
 */
public class AccountsListImpl {
    // static constant array size goes here

    // data fields go here (account array and number of accounts)

    // constructor without parameters (no comments necessary) goes here
    //   (instantiates account array object and initializes number of accts)

    // method to add cell phone account object to account array goes here
          (with comments and parameter tag)

    // method to read data from file goes here
           (-includes comments with parameter and return tags
            -creates a cell phone account object for each line of data and
            -calls previous method to add the object to the account array)

    // method to calculate total bills for all objects in array goes here

    // three methods to calculate the array average, low, and high go here
            (includes comments with return tags)

    // method to create the billing report goes here
}
```

```
/**
 * Description of program, and author and version tags
*/
public class BillingAnalysis {

    public static void main(String[] args) {
        // code to define variables & display program description

        // Loop:
            // code to instantiate AccountsListImpl object
            // Loop:
                  // code to read filename and try to read data from the file
            // code to call other methods
            // code to ask whether to repeat program
    }

    // method to display a billing summary (with comments and parameter tag)
}
```