Personal Finances

1 Front Matter

1.1 Collaboration Statement

The last page of the course syllabus contains a statement of the department collaboration policy with regard to projects. Violation of the collaboration policy can result in failure of the course.

1.2 Reminder: Project Hand in Rules

Read and follow the instructions!

1. All project due dates are specified on Brightspace.
2. All projects handed in must be NetBeans projects.
3. Each student starts the semester with 7 free late days. When handing in a project late you must add a comment telling me that you are using some number of late days for the project.
4. If you hand in a project late, you are not allowed to correct and resubmit. This is the only penalty for late submissions.
5. For projects, you will hand in what is specified by individual assignments. Most of the time it will be an entire zipped project folder. Name the archive with ONLY YOUR LAST NAME. Example: Smith.zip ONLY zip archives will be accepted.

1.3 Objectives

1. Create a Class to hold the data.
2. Create an array to hold the objects of that class
3. Do some searching and summing and summarize the data.

4. Output all requested information with meaningful labels in an understandable format.
5. Use formatted output.

1.4 Grading Criteria

Each group of requirements below is graded as all or nothing. Grading for this project will be done as follows:

Grade Earned	Requirements
D	The main method is the only static method in the class. The main method checks the command line arguments. Usage message is correct for the number of required arguments. The main method in the driver creates an object of the class and calls a run() method passing the arguments. All code handed in is syntactically correct and correctly formatted. The program compiles and runs.
C	Everything above Plus: Create a class to hold the data Declare an array to hold the objects Read and sort the data Correctly sorts the transactions by category. Correctly calculates and displays subtotals for each category. All methods have complete JavaDoc
B	Everything above plus: Correctly calculates the total of all credits and debits and displays those at the end of the category output. Correctly re-sorts the transactions by check number. Correctly displays each transaction as a series of formatted Strings. All methods have complete JavaDoc
A	Everything above plus: Correctly displays the running balance along with each transaction. Correctly calculates the savings balance and a new balance forward at the end of the year. All methods have complete JavaDoc

2 Introduction

Knowing how to handle money and how to budget are necessary life skills. There are lots of apps to help you do that. SoFi, Mint, Quicken, and more will help you see where your money is going with nice charts and graphs. We are going to write our own personal finance app.

3 Input File Format

The input file is is in comma delimited format (csv) Here are the first few lines:

```
Check#,Date,Description,Amount,Type,Category
1836,12/18/2020,68.10,debit,Groceries
1688,4/15/2020,122.64,debit,Entertainment
1632,1/15/2020,1730.77,credit,Paycheck
```

The first line should be read into an array (length 6) of Strings called columnHeadings or headings or something else appropriate.

4 Your Tasks

Remember to follow all of the coding conventions including the order of the methods.

1. Create a class to hold a single transaction. Its private properties are an integer that is the check number, a Date, a description, a double that is the amount, a string that is the type of transaction, and a String for the category the transaction belongs to. The class only requires a parameterized constructor, accessors (getters) for the properties, and a toString() which is simply this:

   ```
   public String toString(){
     return String.format("%7d %-10s %-30s %15.2f %-7s %s",
                          checkNumber, date.toString(), description,
                          amount, type, category);
   ```

   The longest category is 13 characters in length which means that this string is 87 characters long. You might need that information later.

2. The driver will need 2 properties: an array of Transaction objects and an actual count of the number of record read in. Declare the array but don't create it here. The primitive can be set to an initial value of zero.

3. Write the main() method as shown in the Bookstore lab. Check for 2 arguments. "Usage: prog inputFile balance"

4. Write a run() method that accepts the array of arguments.

5. Write a readFile() method that accepts a file name. Use the Bookstore lab as a model. create the array property here and assume that there are no more than 250 transactions in the file. You will need to use a special delimiter to read the file correctly.
   `inFile.useDelimiter("[/, \t\n\r]+");`
   specifies that forward slashes and commas are used as delimiters. After that statement you will be able to use next() and nextInt() as usual. Both the description and the category can have multiple words. In the file they are separated with underscores. When reading those in, you will want to replace the underscores with spaces instead. The input statement will look like this:
   `strVar = inFile.next().replaceAll('_',' ');`

6. Add the call in run() to the readFile() method and test it. It should report 218 records read in.

7. Write a method that sorts the array by the category using a modified version of the SelectionSort() method shown below. Call the method sortByCategory(). It will not need to have an array passed to it since it will sort the array of objects. Modify the swap method so that it only receives the indexes and swaps those two locations in your array of objects.

8. Write a method that calls sortByCategory() and returns a String. The method should calculate a total for each category of items. Use a StringBuilder to put the returned string together. See the output below.

9. Add a call in run() to the method you created above that simply prints the return string.

10. Modify the method that prints the categories so that it totals the debits and credits and displays those at the end of the category totals. See the output below.

11. Write a method that sorts the array by checknumber using a modified version of the SelectionSort() method shown below. Call the method sortByCheckNumber(). It will not need the array passed to it and it can use the same swap method that you already wrote.

12. Write a method that calls the sortByCheckNumber() and returns a string that is entire array of transactions one transaction per line. Use a StringBuilder to put the returned string together.

13. Add a call in run() to the method from above that simply prints the return string. Test this.

14. Modify the method that prints by check number so that it accepts a double that is the starting balance in the checkbook. Modify the output string so that a running balance is printed after each transaction. Debits are subtracted from the balance, credits are added.

15. Modify the call in run to pass the second argument to that method. Since the arguments are Strings you must manually convert it to a double in the call. It should look something like this:
    `System.out.println(printRegister(Double.parseDouble(args[1])));`

16. Modify the method that prints by check number so that 10% of each credit is added to a savings account before the rest is added to the checking account balance.

17. Modify the method once again so that at the end of the year any amount above the original balance is subtracted from the checking account balance and added to the savings account balance before those totals are displayed.

4.1 The Selection Sort Algorithm

Here is the Java code for a sorting algorithm that sorts an array of integers. You will not need to pass an array since you will be sorting the array of objects that is a property of the driver class. Remember that you can't use relational operators for comparison of Strings. The relation will be
`str1.compareTo(str2) < 0`
also remember that you can't use the length of the array, you must use the property that is the actual count of elements.

```
/**
  * Selection Sort finds the smallest element and puts it in first element
  * of the unsorted array. Decreases the size of the array to search at
  * each iteration.
  *
  * @param a the integer array
  */
public void selectionSort(int[] a) {

   int i, j, iMin;
   /* find the min element in the unsorted a[j .. n-1] */
   for (j = 0; j < a.length - 1; j++) {
      /* assume the min is the first element initially */
      iMin = j;
      /* test against elements after j to find the smallest */
      for (i = j + 1; i < a.length; i++) {
         /* if this element is less, then it is the new minimum */
         if (a[i] < a[iMin]) {
             /* hang on to that index */
             iMin = i;
         }
      }
      if (iMin != j) {
          swap(j, iMin);
      }
  }
}
```

4.2 the swap() method

The parameters i and j remain integers, but temp needs to be an object in your array so change the data type accordingly.

```
/**
  * Almost all sorts use a swap algorithm, so a swap method saves lots of
  * lines of code! This method accepts an array and the indeces of the
```

```
  * elements to be swapped and swaps them.
  *
  * @param list an integer array
  * @param i the index of one element to be moved
  * @param j the index of the other element to be moved
  */
public void swap(int[] list, int i, int j) {
   int temp = list[i];
   list[i] = list[j];
   list[j] = temp;
}
```

Your run method for this project will be quite short. It should look similar to this:

```
public void run(String[] args) {
    readFile(args[0]);
    System.out.println(printCategorySubTotals());
    System.out.println(transactions(Double.parseDouble(args[1])));
}
```

the method printCategorySubTotals() should call the selectionSortByCategory() method before doing anything else, and printTransactions() should call the selectionSortByCheckNumber() method before doing it's job.

5 Output Examples

```
Check#    Date       Description                  Amount
  // lots of lines missing here
  1795    10/15/2020 Toyota Financial             359.29
  1811    11/15/2020 Toyota Financial             359.29
  1830    12/15/2020 Toyota Financial             359.29
Subtotal for category Car Loan                          4,311.48
  // lots of lines missing here
Total Debits:   39,254.15  Total Credits:   45,000.02
```

If the balance parameter value is 1000.00 this would be part of the output for the printTransactions(double balanceForward)

```
Check#  Date       Description                 Amount  Type    Category      Balance Forward
                                                                                  1,000.00
  1623  1/1/2020   TechnoCompany              1,730.77 credit  Paycheck        2,557.69
  1624  1/1/2020   Johnson Property Mgmt      1,600.00 debit   Rent              957.69
  1625  1/2/2020   CMP                          164.57 debit   Utilities         793.12

    // 215 lines missing here
Savings Balance   Adj Checking Balance
    5,745.87             1,000.00
```

Adding commas to numbers that are output is easy. Just put a comma between the percent sign and the column width number like this: `%,15.2`

6 Complete the Documentation

Generate JavaDoc Documentation and complete it. Choose Tools - Analyze JavaDoc. In the window at the bottom click the check box for the driver and click the button that says Fix Selected. This will create part of the documentation. You must then go back through each file adding file descriptions, and completing the documentation for each method.

I actually read your documentation. This is easy. It's not a term paper and it is not creative writing. Pay attention to the video on documentation.

7 PROGRAMMING STYLE

Your programs will be evaluated for formatting, use of meaningful identifiers, and documentation.

Although no points are allocated for that on this assignment, somewhere between twenty and thirty percent of your grade will derive from the programming style on this and all future assignments. Note, the NetBeans environment can take care of formatting for you and it can generate skeleton JavaDoc which you MUST complete. Up to 30 points can be deducted for failure to adhere to the coding conventions, failure to correctly format your code, and failure to complete the documentation.

Formatting is a question of clicking two times. Choose Source ==>Format. It is the first thing I do when I open your source file. If nothing changes, you did it already. NetBeans does not automatically wrap your lines. You have to do that. Break long statements at appropriate places so that they don't go beyond the line on the right of the editor window.