CSCI 3901 Assignment 1

Due date: 11:59pm Friday, September 13, 2019 in Brightspace

Problem 1

Goal

Get practice in decomposing a problem, creating a design for a program, and implementing and testing a program.

Background

Matrices of data appear commonly in data analysis tasks. Each row represents some experiment or individual and the columns represent attributes or data that we are gathering about the experiment or individual. Often, we want to apply some transformations to the data in the columns of the matrix before we start our analyses.

The ability to interact with the data as we transform it and to see the changes allows an experimenter to guide the data transformations and to ensure that the outcomes are what is expected. Today, we might use a statistical analysis software package or a spreadsheet to make these transformations.

In this assignment, we will create a tool to do simple, interactive transformations to this kind of data.

Problem

Write a class that stores a matrix of data and can have a set of operations invoked on the data. I will provide a "main" program that will then let a user invoke the operations on the data.

The normal use of your class will have the following sequence:

- Read data from a file
- Do any of the following, in any order, a number of times
    - Create new columns for data
    - Set a column's data values based on a restricted form of equation (more on the restriction later)
    - Show the top 5 rows of data
    - Show all of the data
    - Write the data to a file

Each column of data in the incoming data file will have a column name. References to columns will be by the column names.

The equations that can change the value of a column will have at most one binary operator. The two operands of the operator will be either a constant or a column name. For example, if we have columns called “baseprice”, “price”, “quantity”, “tax”, and “total” then we might have any of the following equations:

- tax = price * .15
- total = price + tax
- quantity = 1
- price = baseprice

We would _not_ have equations like

- total = (price + tax)
- total = baseprice * quantity * .15

The only binary operators that will be used are +, -, *, and /. More specifically, all equations will have one of the following forms:

- <columnName> = <value> <operator> <value>
- <columnName> = <value>

Where <columnName> is a column name, <value> is either an integer or a column name, and <operator> is one of + (addition), - (subtraction), * (multiplication), or / (division).

We will not add rows to the data interactively.

Your class will be called “DataTransformer” and will include the following methods:

- boolean clear() – erase all data from your object. Return True if the object is empty after the operation.
- Integer read(String filename) – read in the contents of the file to the object. Return the number of data rows read. The data should replace any information already in the class. Columns will be separated by tab characters.
- boolean newColumn(String columnName) – create a new column with the given name. The column will only store an integer and will begin with a value of 0. Return True if a new column was created and is ready to be used.
- Integer calculate(String equation) – apply the given equation to the data in the object and return the number of rows that were calculated for the equation. If a calculated value will have decimal places then round the value to the nearest integer (a value of .5 rounds up).
- void top() – print the first 5 rows to the screen. Include a starting row with the column names. Separate the columns with a tab character. The order of the columns should be the same as in the input file, with new columns appearing afterward in the order of creation.
- void print() – print all rows to the screen. . Include a starting row with the column names. Separate the columns with a tab character. The order of the columns should be the same as in the input file, with new columns appearing afterward in the order of creation.

- Integer write(String filename) – write the contents of the object to the given file. Include a first row with the names of the columns. Separate the columns with a tab character. Return the number of data rows written to the file.

Inputs

An input data file for your class will have all columns separated by a tab character. The first line of the file will have the column names. Every line after that will contain data for your object.

Each data entry in the file will be either an alphabetic string (possibly with spaces) or an integer.

Example:

name	baseprice	quantity
bottled water	5	3
bread	2	5
rice	3	3

Sample sequence of transformations

read: (filename of the file shown in "Inputs" section)
newColumn: value
newColumn: tax
newColumn: total
calculate: value = baseprice * quantity
calculate: tax = value * 15
calculate: tax = tax / 15
calculate: total = value + tax
print:

Output of print would then be:

name	baseprice	quantity	value	tax	total
bottled water	5	3	15	2	17
bread	2	5	10	2	12
rice	3	3	9	1	10

Assumptions

You may assume that

- The input file will have at most 10 columns.
- File names will contain the full path to the file, including any file name extensions.
- No line in the file is more than 80 characters.
- Column names will be a single alphabetic string. They will not have any spaces or non-letter characters.

- The user will not ask you to make a calculation that divides by 0.
- String columns will only contain alphabetic characters.
- Data in a column in a data file will always have the same data type (you won't get a column where row 1 has an integer in the column and row 2 has an integer in the column).
- There will always be at least one space between each component of an equation.
- Every line in an input file will have the same number of columns of data.

Constraints

- Write your solution in Java. You are allowed to use data structures from the Java Collection Framework.
- Your program should treat all column names in equations as being case insensitive.
- If in doubt for testing, I will be running your program on bluenose.cs.dal.ca. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- You will be provided with a "main" class on the course brightspace page by Monday morning that will give you an interface through which to invoke your class.
- You are permitted to create more than one class in support of your work.
- I recommend thinking about how you want to store your data first. Consider some of the built-in data types that do not worry about the size of the data. Next, design, code, and test functionality one method at a time. Do the "calculate" method last.
- Use your design notes to begin the external and internal documentation for your program.
- Review the "lessons learned" slides from the course notes.
- Your methods will be invoked with error conditions. You are expected to handle these errors.
- Review the test cases to identify special cases that you will want to consider in your work.
- Your class should never end by producing a Java error stack trace.
- The marking scheme has no marks for time or space efficiency.
- Use the number of marks for the methods and the set of test cases as guides for how much time to devote to each method.
- Recognize when you are spending time with no progress and ask for help before you find yourself spending hours with no progress to show for the time.

Marking scheme

- Documentation (internal and external) – 3 marks
- Program organization, clarity, modularity, style – 5 marks
- Ability to process file contents (read, write) – 6 marks
- Ability to add columns – 2 marks
- Ability to print data – 4 marks

- Ability to change data in columns – 10 marks

The majority of the functional testing will be done with an automated script, so stick to the input and output formats.

Test cases

Generic file processing (read, write):

- Null as the file name
- Empty file name
- File name that doesn't exist
- File with no lines in it
- File with column titles only in it
- File with 1 line of data (plus column titles) in it
- File with many lines (so a middle line)

Specific file processing (read, write):

- File with 1 column of data
- File with 2 columns of data
- File with 10 columns of data
- File with data lines below 80 characters
- File with data lines at 80 characters
- File with one column name being 80 characters (just one column in the file)
- File with a string column
- File with an integer column
- Load data into a new object
- Load data into an object that already has data in it

Clear:

- Invoke on a new object, even before any data has been loaded
- Invoke on an object with data in it
- Invoke on an object with columns defined but no rows of data
- Invoke on an object twice in a row (i.e. on an already-cleared object)

newColumn:

- Invoke with an empty string
- Invoke with a null for the string
- Invoke on a new object, before any data has been loaded
- Invoke on an object that has 1 column defined
- Invoke on an object with 9 columns defined
- Invoke on an object with 10 columns defined
- Invoke with a column name all lower case
- Invoke with a column name all upper case
- Invoke with a new column name

- Invoke with a column name that already exists (same upper/lower case)
- Invoke with a column name that already exists (different upper/lower case)
- Invoke with a column name that includes one or more spaces

Calculate:

- Invoke with an empty string
- Invoke with a null for the string
- Invoke with an object with no data loaded
- Invoke with an object with one column defined and no data rows
- Invoke with an object with one column defined and 1 row of data
- Invoke with an object defined with one column defined a many rows of data
- Invoke with an object with several columns defined (changing the number of rows of data)
- Invoke to store in a column that was created from the file contents
- Invoke to store in a column that was created with newColumn
- Invoke with an equation that has no = sign
- Invoke with an equation of the form “a = b” where
    - “a” is an existing column name and “b” is a positive integer
    - “a” is an existing column name and “b” is a negative integer
    - “a” is an existing column name and “b” is an existing column name
    - “a” is an existing column name and “b” is a column name that doesn’t exist
    - “a” is a column name that doesn’t exist
- Invoke with an equation of the form “a = b c d” (“c” as an operator) where
    - “a” is an existing column name, “b” and “d” are integers, and “c” is one of the operators +, -, *, and /
    - “a” is an existing column name, “b” is an integer, “d” is an existing column name, and “c” is one of the operators +, -, *, and /
    - “a” is an existing column name, “b” is an integer, “d” is a column name that doesn’t exist, and “c” is one of the operators +, -, *, and /
    - “a” is an existing column name, “d” is an integer, “b” is an existing column name, and “c” is one of the operators +, -, *, and /
    - “a” is an existing column name, “d” is an integer, “b” is a column name that doesn’t exist, and “c” is one of the operators +, -, *, and /
    - “a” is an existing column name, “b” and “d” are existing column names, and “c” is one of the operators +, -, *, and /
    - A valid equation except for having “c” as an unrecognized operator, like %
    - A valid equation with “b” as a negative integer
    - A valid equation with “d” as a negative integer
    - A valid equation with a division by 0
    - A valid equation where “a” and one of “b” and “d” is the same column name as “a”
    - Invoke with a valid formula and a varying number of spaces between elements
    - Invoke with a valid formula that uses a string column
- Invoke with a formulate that rounds up for one row and rounds down for another.

Printing (first, print, and write):

- Print an object with no columns
- Print an object with 1 column of data
- Print an object with 2 columns of data
- Print an object with 1 row of data
- Print an object with 5 rows of data
- Print an object with 6 or more rows of data
- Print an object with a “newColumn” column that has not been assigned data to it

Writing:

- Write an object with 10 columns of data
- Write an object with 11 columns of data