ArrayLists and Data

Haverford CS 106 - Introduction to Data Structures

Lab 2 (one week)

1 Note

- Recall the difference between `==` and `.equals()`. If you aren't sure, discuss with a peer or a TA/instructor!
- The terms "Caucasian" and "White" are used interchangably (at least in the U.S.). The dataset uses them as synonyms.
- As always, make sure your `main` method does not throw any Exceptions – it should be handling them.
- As always, please check your work with the `Verify` files! Do not make changes to these files; instead, make corrections to your work!

2 Finishing Your Data Storage

In the previous part of this lab assignment you created a class to store a single row of the dataset. In this lab, we will create a class to store the full dataset.

As always, make sure you comment thoroughly throughout the lab.

2.1 Create a new class

Create a class that will represent the entire data set (not just one row). Think carefully about what to name it, based on your understanding of the data, so that it correctly indicates what data is being stored. Meaningful names are an essential part of good programming practice!

2.2 The ArrayList field

We will store each row in an ArrayList. The ArrayList should hold the objects you made in lab 1. Make a field in your dataset class that holds an ArrayList, and make sure the datatype of the ArrayList is the object type you created in the last lab.

2.3 Methods

2.3.1 Constructor

Add a constructor that initializes the field you created as a new ArrayList. It should not take any parameters.

You should also write necessary getter and setter methods. Note that you might add more as necessary as you go along.

2.3.2 Adding to the array list

Recall that the data is read into an ArrayList of String arrays (see pre-lab). We need to iterate through the dataa ArrayList, convert each row (String array) of the data ArrayList into the object you made in the last lab, and add the object to the ArrayList in this class.

Write a method that implements or helps to implement adding to the ArrayList in the class you made. You may choose to design it how you see fit, but feel free to refer back to the pre-lab for ideas.

Since exceptions may occur due to invalid or unexpected input, you will want to have try-catch statement to handle them.

2.3.3 Test your addition method

In your `Main` class, make a method called `testAdding()` to test the method you just wrote. If you haven't implemented `toString()` method in the last lab, you may find it beneficial to do that now.

You might use JUnit as done in the last lab or print to check the list.

3 Reading in Data

3.1 About the Opencsv library

Opencsv is a library that allows you to read in data. You can read in data from "compas-scores.csv" using the below code. You'll also need to add the appropriate imports - Eclipse can help you find them.

We will use CSVReaderHeaderAware from the OpenCSV library for this lab. Because we are using HeaderAware reader, it understands that the first row is the "header" and does not include it in the data.

The following code demonstrates how you should use it to read the data from "compas-scores.csv":

```
CSVReaderHeaderAware reader = new CSVReaderHeaderAware(
                                   new FileReader("compas-scores.csv"));
ArrayList<String[]> myEntries = new ArrayList<String[]>(reader.readAll());
reader.close();
```

This will give you an ArrayList (`myEntries`), where each index is a String array holding the row data. The indices in the String array correspond to the column indices. For example, the following is a visualiazation `myEntries`:

```
<  ["Male", "Other", "F", ...],               // row 1
   ["Male", "African-American", "F", ...],     // row 2
   .... >
```

3.2 Add the data

After reading in the data, use it to populate your data structure in the class you wrote. Answer the following questions in your `README.md` file (add a section below your name and above the questions about difficulty/timing):

1. If any of your validation methods indicate that your precondition assumptions were not met by the data, document what preconditions were violated and in what way. Update your validation methods one error at a time, documenting all issues, until you can read in all the data. If this did not happen, mention that.

2. Describe a person who would generate data that would *not* pass your (updated) precondition assumptions.

As a last resort after updating your pre-conditions, you should make sure you skip any rows that cause errors so that the data processing continues. *Hint: if you're having trouble with this, where could you put the try-catch statement so that the "trying" happens for each row?*

4 Replicate the Analysis

Now that you have the data read into your data structure, we can reproduce the analysis ProPublica shows in their chart "Prediction Fails Differently for Black Defendants."

4.1 Write the relevant methods

Recall how each percentage in the table was calculated. Read ”Lab 1: Understanding Propublica.pdf,” and discuss with a peer or an instructor if you need help.

In the data storage you created, write method(s) that iterate through the ArrayList of objects you created in lab 1 and calculate the percentages (in decimals). They should return the percentages as decimals (`doubles`).

4.2 Call the methods

Call these methods in your `Main` class's `main` method. You can check your work by making sure your numbers match the numbers in the ProPublica's chart.

Take a look at `PropublicaDataTable` class (included with the starter files) - specifically its constructor. Now that you know what you need to construct an instance of the table, make a `PropublicaDataTable` object by passing your numbers into the constructor.

Make sure you set the `racialBiasTable` instance variable to the correct table in the `main` method (+/- 0.1 percent is okay). Feel free to print out the table. Make sure you are able to run `VerifyFormat_Lab2.java` and get a passing status.

5 Your Custom Analysis

What you have already done should not be altered by your work in this section. This means you should write **additional** methods instead of changing the methods you wrote so that your original methods stay the same.

You may especially want to commit and push the work you have done so far as a checkpoint you can revert back to.

5.1 What counts as recidivism?

Find charges that, based on the `r_charge_desc`, you think should not count as recidivism. Perform an additional analysis with these decisions - you may need to write additional methods. Do this using an optionally run method so that you can still run the previous version of the analysis. Describe the choices you made and what the resulting analysis shows in your `README.md`.