Data Structure Design

ProPublica Lab - Part 1 (two weeks)

1 Designing a Data Structure

1.1 Understanding the Data

When designing any data structure, we first need to understand the details of the data that we will store and what types of queries we may want to make about that data. For this lab, the data we'll be working with comes from a ProPublica story about a risk assessment tool called COMPAS.

To understand the data and the larger context, look at these sources in order:

1. ProPublica's introductory video: https://youtu.be/17eDz5HA_qI
2. Article "Machine Bias" by Propublica: https://www.propublica.org/article/machine-bias-risk-assessmentsin-criminal-sentencing
3. Handout "Lab 1: Understanding Propublica.pdf"
4. Video: "Context of Risk Assessment Tools"
5. Look at the (cleaned) data in "compas-scores.csv".

Feel free to discuss the content with peers and/or instructors.

1.2 Creating a Class Representing a Single Row

1.2.1 Create a class

Think about what a single row in the CSV represents. Create a class that holds information about a single row. Be sure to name it meaningfully based on your understanding of what the single row represents. Feel free to discuss with your peers or instructors what a meaningful name should be.

Using javadoc style, write a comment to describe the class - what it represents, what contents it holds, etc.

1.2.2 Fields and methods

Add necessary fields to represent the information about what the class represents based on the CSV - think about its columns! Think carefully about what the most appropriate data type of these fields should be; you should include `enum`s as a possibility but make sure they are an appropriate data type - e.g. you should not make everything be `enum` or `String`.

After you have created the fields, add any necessary getter and setter methods to the class.

Be sure to use appropriate coding style and add appropriate Javadoc comments.

1.2.3 Constructor

We now need to make the class accept data from outside to initialize the fields you created in Section 1.2.2. Make a constructor that takes in multiple variables (e.g. `String sex`, `String race`, `...`) and initializes the fields of your class.

1.2.4 Test your class

Testing is an important part of developing data structures. In the `Main` class, make a `testObject()` method that makes an instance of the class and tests your getter and setter methods.

Call your `testConstructor()` method in your `main` method of your `Main` class to test your work. For example, your class should look similar to the following, where *ClassName* is the name of the class you created in this section:

```
import org.junit.Assert;

public class Main {
    public static void main(String[] args) {
        testConstructor();
    }

    public static void testConstructor() {
        // construct an object
        ClassName meaningfulName = new ClassName(String sex, ...);

        // test the getter method for the sex field
        // assertEquals will return no output if the Strings are equal
        // otherwise, it will throw an Exception
        Assert.assertEquals(sex, meaningfulName.getSex())
```

```
        // other tests here
    }
}
```

You can also test your work using `System.out.println` to print variables, but if you do so please declutter the console output by commenting out these calls once you're sure it works.

Continue to test your work in the following section similar to how you did here.

1.3 Getting ready to Replicate the Analysis

The goal of this lab is to get ready to replicate ProPublica's analysis shown in the table *Prediction Fails Differently for Black Defendants*. In the next lab, you'll do the actual replication. To prepare the data structures appropriately and understand the analysis you'll be working towards, make sure you have read the article "Machine Bias" by ProPublica and the handout "Lab 1: Understanding Propublica.pdf".

In order to do this, we need to determine if the person represented by the row fits the description in the table. For example, to see if someone is part of the 25.3% in the table, we need to check

1. if the person is white (or Caucasian)
2. if the person did not re-offend (i.e., was not labeled as being rearrested within 2 years)
3. if the person was labelled "high risk"

More generally, we need to check the person's race (white or Black), whether they re-offended, and whether they were labelled "low risk" or "high risk" of recidivism. Make these methods that return `true` or `false` depending on whether a person fits this criteria: `isWhite()`, `isBlack()`, `hasReoffended()`, `isLowRisk()`, and `isHighRisk()`.

You will use these methods in the next lab to determine things like (from "Lab 1: Understanding Propublica")

```
out of only the white defendants who did not reoffend,
23.5% of them were labeled as "high risk"
```

to calculate whether a person should be counted for that 23.5% of white people who did not re-offend, but were labelled "high risk". Since this lab deals with only one row of the data, we will not be able to replicate the chart yet.

Make sure you follow appropriate coding practices, update and maintain appropriate comments about the class and the methods, and test your work. Please name your testing method `testBools()` and follow instructions as described in Section 1.2.4.

2 Data Validation

To help us read the data from the CSV into our data structure in the next lab, we need to make some modifications. Specifically, we need to make a new constructor and put safe guards in place about what can be passed into the constructor.

2.1 New Constructor

In the next lab, we will read each row of the CSV as String arrays.

Do not get rid of the original constructor. In Java, we can "overload" methods - meaning the methods can have the same name as long as their parameters are different. Java compiler will *automatically* look for the method that fits the one you're calling.

Create a second constructor for the class that takes in a String array and initializes the fields. The indices of the array will correspond to the column of the row.

Please follow testing descriptions from Section 1.2.4. In testing your work, please name your method `testStringArrConstructor()`. To simulate testing, you can use the following syntax to create a new String array:

```
public static void testStringArrConstructor() {
    String[] row1 = new String[] {"Male", "Other", "F",
        "Aggravated Assault w/Firearm", "1", "Low", "0", "", ""};
    ClassName meaningfulName = new ClassName(row1);

    // more code to write for testing
}
```

2.2 Safe-guarding the constructor

We should put some safeguards in place to make sure that we don't have any data errors when we do the imports from the CSV.

2.2.1 What are the valid values?

Determine what the valid options are for each field in your class - you'll likely want to look at "compas-scores.csv" again. Use Javadoc style to document these preconditions.

2.2.2 Are the values valid?

Within the constructor, perform a check before initializing the fields to ensure the fields only hold valid values that you just determined. Invalid values should throw an exception.

Make sure that tests you do deals with exceptions - surround your test method in try-catch block. This applies for *all* tests you've written for this lab. Remember that your Main class should never throw an exception - it should be handling them!

2.2.3 Discuss!

Talk to someone else in the class and discuss what pre-conditions they decided for each of their fields. How do they compare to yours? Feel free to update any pre-conditions in the comments and (if any) document any potential disagreements with the people you talked to.

2.2.4 Test!

You're a good programmer, so you test what you do as you go. Check that your pre-conditions work by passing a String array with valid values and some with invalid values.

2.3 Optional: `toString()` and `equals()` methods

If you try to print an object, Java will output the memory address of the object. However, this doesn't tell us much and makes it difficult to see what the object is like. To make meaningful print statements, override the `toString()` method to show some of the fields. You can do this by simply creating a method in the class with the precise signature: `public String toString()`

Similarly, the `equals()` method, used by `assertEquals` to determine equality between objects, can be overridden for the new object you have created. Create a method in the class with the signature: `public boolean equals(Object o)`. The method should return `true` if the objects are equal and `false` if not. You will need to cast the given object to the same type as your created task before you can check equality of the individual fields.