CSCI 1933 Lab 4

IntelliJ Intro, OOP, File I/O, and Databases

Lab Rules

You may work individually or with *one* partner in lab. We suggest that you have a TA evaluate your progress on each milestone before you move onto the next. The labs are designed to be completable by the end of lab. If you are unable to complete all of the milestones by the end of lab, you have **until the last office hours on Monday** to get those checked off by **any** of the course TAs. We suggest you get your milestones checked off as soon as you complete them since Monday office hours tend to become crowded. If you worked with a partner, both of you must be present when getting checked off to receive credit. You will only receive credit for the milestones you have checked off by a TA. There is nothing to submit to Canvas for this lab.

Attendance

Per the syllabus, students with 4 or more unexcused lab absences over the course of the semester will automatically fail the course. The TAs will take attendance during lab; it is expected that students will be actively working on the lab material. Your physical presence in lab is not sufficient to be marked as present for the purposes of attendance. *Your physical presence in lab is not sufficient to be marked as present for the purposes of attendance.*

Setting up IntelliJ

If you are already comfortable using IntelliJ, you may skip this section, although it is recommended to ensure you are following best practices.

You've already received your first project in the class. When creating these larger code bases, it can be tedious to switch back and forth from the text editor to the terminal and in between all the code files. This is where an Integrated Development Environment (IDE) comes in handy.

We recommend using IntelliJ IDEA by Jetbrains for this class. IntelliJ is an industry standard that is free for students, you may apply here to get your free student license or use the community version.

Note: Included in the lab is a PDF explaining the initial setup in more depth.

Now that IntelliJ is downloaded and you have your license registered, you can create a new project from scratch. IntelliJ will find your Java SDK and ask if you want any external libraries. For this lab, we will use native Java, so just click next. Rename the project name to `lab4` and change the project location to your `labs/lab4` directory.

Paste the provided `.java` files into the new `src/` folder, and the `.csv` files into `lab4/`.

Now you have all your files in place, but IntelliJ still needs to know which `main` to run. In the top right corner, there is a button to add a new configuration. Select that then select the `Application` option, your main class will be `Owl` for Milestone 1 and `OwlPopulation` for the rest. You may add multiple configurations and switch between them on the fly using this button as well. Now that IntelliJ knows where your `main` is, hitting the green hammer and green play button will build and run your project respectively.

Tip: Try running with Owl as the main, you will see an output message if your IntelliJ is correctly set up.

Analyzing Owl Dataset

1 Owl Class

One of the tracks offered by the Computer Science degree is Big Data. An important part of Big Data is being able to process a large dataset and perform some sort of analysis on it. For this lab, we're going to be doing some statistical analysis on populations of owls. Before we do so, we need to model our owls: let's make an `Owl` class.

Each `Owl` has its own `name`, `age`, and `weight`, so we need class variables (types `String`, `int`, and `double` respectively) to reflect this. The first method we need to make is a constructor for the `Owl` class:

- `public Owl(String name, int age, double weight)` — this initializes the class variables of each `Owl` object to the arguments passed in.

It is good practice to make all your class variables `private`. Once we do this, we also need to make getter and setter methods to access these private variables from outside the `Owl` class.

Finally include the method:

- `public boolean equals(Owl other)` — This returns `true` if the name, age, and weight of the `Owl` are equal to `other`'s name, age, and weight. Otherwise, this returns `false`. Remember to use the `equals` method to compare `Object` types like `String`, and use `==` to compare scalar types such as `double` and `int`

Milestone 1:
In a main method, instantiate two owls, and demonstrate your `equals` method working.

2 OwlPopulation Class

Next, we'll need to create an Object type to model a population of owls. For this milestone, modify the given `OwlPopulation` class. This class will have the following variables: `private String fileName` and `private Owl[] data`. The `OwlPopulation` constructor should take in a filename and call the following method.

- `public boolean populateData()` — this will readin the CSV file and construct the Owl objects to put into the OwlPopulation's `data` attribute.

In order to read in the CSV file, we will again utilize the `Scanner` class. The `populateData` method is partially filled out for you. It already reads in a file line by line. You may modify this code to use in your `OwlPopulation` class.

Hint: A CSV file (comma-separated-value file), as its name suggests, separates each value with a comma. In this file, each line contains the [name, age, weight] attributes for an individual owl.
The following methods may be useful to you in parsing the file:

- `split(String delimiter)` is called on a String. It splits the string into an array of substrings delimited by the argument passed in.
- `Integer.parseInt(String num)` and `Double.parseDouble(String num)` both take in a `String` object and return an `int` or javadouble version of the String input.

Milestone 2:
Show `OwlPopulation` being constructed correctly. Print the number of owls in the population to the console.

3 Taking Statistics on an OwlPopulation

Make the following methods to collect statistics on the OwlPopulation:

- `public double averageAge()` — calculate and return the average age of all the Owls in the population.

  Note: This function returns a `double`, while an `Owl`'s `age` attribute is of type `int`.

- `public Owl getYoungest()` — returns the youngest `Owl` in the `OwlPopulation`. If no owl is found, return `null`.
- `public Owl getHeaviest()` — returns the heaviest `Owl` in the `OwlPopulation`. If no owl is found, return `null`.

- `public String toString()` — this will return a summary of the population. The summary should include the name and age of the youngest `Owl`, the name and age of the heaviest `Owl`, and the average age of the owls in the population. The format of the summary is up to you.

Milestone 3:
Make a main method in your `OwlPopulation` class and demonstrate your statistic collection working — construct `OwlPopulation` objects from the two files supplied (owlPopulation1.csv and owlPopulation2.csv) and call `toString()` on each.

Reflection:
What are the time complexities of `averageAge()`, `getYoungest()`, and `getHeaviest()`?

4 Merging OwlPopulations

Write the method in the `OwlPopulation` class `public void merge(OwlPopulation other)` which takes in an `OwlPopulation` and merges it with the population it is called on. Take care to not include "duplicate owls" in your population.

Hint: The `equals()` method written for the first milestone may be useful in implementing the helper `containsOwl(Owl other)` function.

Milestone 4:
In your main method, merge two `OwlPopulations` and print the new statistics of the merged population to the console. Where do the original populations go? Where is the "merged" population stored?

Reflection:
Assuming that the size of the data set is n (the total number of owls in the two databases combined), try to quantify the time complexity of your merge() algorithm in terms of n. You may wish to approach this by comparing to the three sort methods given in lecture which are all $n^2 complexity$.