Sprint 3
Refactor and Reuse: Towards Sustainable Code

Collaboration/Plagiarism Policy: This assignment is **individual work**. Sending, receiving, posting, reading, viewing, comparing, or otherwise copying any part of course assignments or solutions is not allowed except when explicitly permitted in assignment instructions. This includes solutions from other students in the course, past or present. See the course syllabus for penalties for collaboration policy violations.

Learning Objectives

- Simplify code structure by implementing inheritance to extend an abstract class.
- Implement the Comparable interface to define a natural ordering of objects.
- Create Comparator classes to compare objects in additional ways.
- Implement unit tests using the JUnit framework.

Grading Rubric

- 70% Methods function properly and your JUnit testing
- 30% Design and readability (See Coding Style Guide)
  - 10% Correct indentation
  - 10% Naming Conventions
  - 10% Well-commented
    - Comment at the head of each file (see "Style" section below)
    - Comment at the head of every major (and new) method
    - General in-line commenting as needed to highlight interesting logic
    - Comment at the head of each JUnit test (brief 1-line is fine)
    - Comment main-method testing code within your `main()` method

Instructions: The project manager noticed the `PhotoLibrary` and `Album` classes have nearly identical functions with regard to a list of photographs and wants you to harness the power of inheritance to clean then up.

Additionally, the project manager is aware that as a photo library grows, it will be helpful to be able to better manage large numbers of photos through sorting, searching, and other techniques. To that end, you have been requested to provide ways to compare `Photograph`s as outlined below.

Harness the power of inheritance

Create a new **abstract** class named `PhotographContainer` that holds `Photographs`. From the `Album` class, move the following fields and methods into the `PhotographContainer` class. Once you have moved them, remove them from the `Album` class. *Note: we've kept the descriptions of each, but you should not need to write any new code.*

- Fields:
  - `name` (**protected**)
    A String containing the PhotographContainer's name in whatever form it was provided.
  - `photos` (**protected**)
    An `ArrayList<Photograph>` of photos in the container. You are required to use `ArrayList<Photograph>`, not an array or other kind of set.
- Methods:
  - `public String getName()`
    The getter written for the `name` field.
  - `public void setName()`
    The setter written for the `name` field.
  - `public ArrayList<Photograph> getPhotos()`
    The getter written for the `photos` field.
  - `public boolean addPhoto(Photograph p)`
    Add the `Photograph p` to the list of the current object's `photos` if and only if it was not already in that list. Return `true` if the `Photograph` was added; return `false` if it was not added. Return `false` if `p` is null;
  - `public boolean hasPhoto(Photograph p)`
    Return `true` if the current object has `p` in its list of `photos`. Otherwise return `false`.
  - `public boolean removePhoto(Photograph p)`
    Remove `Photograph p` from the container, if it exists in the list of photos. If successful, return `true`; else return `false`.
  - `public int numPhotographs()`
    Return the number of Photographs in the current container.
  - `public boolean equals(Object o)`
    Following the standard rules and conventions as shown in class, return `true` if the current container object's name value is equal to the name value of the container object passed to `equals()`. Otherwise, return `false`.
  - `public String toString()`
    Generate a String that has the name of the container on the first line, followed by a list of photo filenames in the `PhotographContainer`.
  - `public int hashCode()`
    The hashcode method we provided for `Album` for Sprint 2.
- Constructor: Write a constructor for `PhotographContainer` that matches that of the `Album` class from Sprint 2. It must take a `String name` as parameter and initialize all other fields of the container object (i.e. the `ArrayList`).

From the `PhotoLibrary` class, move the following fields and methods into the `PhotographContainer` class. Once you have moved them, remove them from the `PhotoLibrary` class.

- Methods:
  - `public ArrayList<Photograph> getPhotos(int rating)`
    Return an `ArrayList<Photorgraph>` of photos from the `photos` list that have a `rating` greater than or equal to the given parameter. If the rating is incorrectly formatted, return `null`. If there are no photos of that rating or higher, return an empty `ArrayList<Photograph>`.
  - `public ArrayList<Photograph> getPhotosInYear(int year)`
    Return an `ArrayList<Photograph>` of photos from the `photos` list that were taken in the year provided. For example, `getPhotosInYear(2018)` would return a list of photos that were taken in 2018. If the year is incorrectly formatted, return `null`. If there are no photos taken that year, return an empty `ArrayList<Photograph>`.
  - `public ArrayList<Photograph> getPhotosInMonth(int month, int year)`
    Return an `ArrayList<Photograph>` of photos from the `photos` list that were taken in the month and year provided. For example, `getPhotosInMonth(7, 2018)` would return a list of photos that were taken in July 2018. If the month or year are incorrectly formatted, return `null`. If there are no photos taken that month, return an empty `ArrayList<Photograph>`.
  - `public ArrayList<Photograph> getPhotosBetween(String beginDate, String endDate)`
    Return an `ArrayList<Photograph>` of photos from the `photos` list that were taken between `beginDate` and `endDate` (inclusive). For example, `getPhotosBetween("2019-01-23", "2019-02-13")` would return a list of photos that were taken in between January 23 and February 13 of 2019. If the begin and end dates are incorrectly formatted (specifically with month not between 1 and 12 or day not between 1 and 31), or `beginDate` is after `endDate`, return `null`. If there are no photos taken during the period, return an empty `ArrayList<Photograph>`.

Remove the following methods and fields from the `PhotoLibrary` class, since they will be inherited by the abstract class `PhotographContainer`:

- Fields:
  - `name`
  - `photos`
- Methods:
  - `getName()`
  - `addPhoto()`
  - `getPhotos()`
  - `hasPhoto()`
  - `numPhotographs()`

Modify your `PhotoLibrary` class to **extend** the abstract class `PhotographContainer`. It will then inherit all the methods available from the abstract class, including the list of photos. Since we inherit `removePhoto`, but it does not work the way we need (i.e. removing photographs from the library's albums as well), override it by renaming the `erasePhoto` method to `removePhoto`. Remember to keep the other methods in `PhotoLibrary` that you had implemented in `HW3`, such as `createAlbum` and `removeAlbum`.

Modify your `Album` class to **extend** the abstract class `PhotographContainer`.

Comparing Photographs

For this section, you are asked to (1) modify your `Photograph` class to implement the Comparable interface, and (2) write **two** `Comparator`s.

1. Modify your `Photograph` class to **implement** the `Comparable` interface. This interface instructs Java on the *natural order* for objects of type `Photograph`. We will define the natural ordering of photographs to be first by their `dateTaken`, and second by their `caption`. That is, if two `Photograph` objects have the same value for `dateTaken`, they should be ordered (alphabetically) by the `caption`. To implement this interface, you must implement one additional method in the `Photograph` class:

   – `public int compareTo(Photograph p)`
   Compares the `dateTaken` of the current `Photograph` object with the parameter `p`. If the current object's `dateTaken` is before `p`'s, return a negative number. If `p`'s is earlier, return a positive number. If they are equal, return the comparison of the this object's `caption` with `p`'s `caption`.

   Hint: Comparing dates of the form "YYYY-MM-DD" can be done with normal lexicographical string comparisons (i.e. alphabetical sort), since comparing strings left-to-right will first compare the year, then the month, then the day.

2a. Create the class `CompareByCaption` that implements the `Comparator` interface and compares two `Photographs` by caption (in alphabetical order). If two captions are identical, then compare by `rating`, in descending order with the highest-rated photo first.

2b. Create the class `CompareByRating` that implements the `Comparator` interface and compares two `Photographs` by `rating` (in descending order). If two ratings are identical, then compare by `caption` in alphabetical order.

Full specifications for the `Comparable` and `Comparator` interfaces are available at:

- https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html

Reminder: For each of the methods, you must write the exact method signature as specified. For your submission to pass the tests, field and method names must match. For example, our test cases will look specifically for the method `getPhotosInYear`. If you rename the method to be `getPhotosinYear` (notice the difference in capitalization) then any tests relating to / involving this method will not pass since it would appear that you do not even have such a method as `getPhotosInYear`. So be very careful to check that you are: (1) using the same names we ask you to use, paying attention to capitalization, and (2) do not change the method return type, number and types of the parameters, or visibility modifiers (private vs public).

We bring this issue to your attention because from past experience we've found that this will reduce any stress associated with debugging your code and interpreting the results given back to you by our test cases.

Testing: You will need to write at least **two** JUnit tests for each of the following methods:

- `removePhoto(Photograph p)` in PhotoLibrary
- `compareTo(Photograph p)` in Photograph

- `compare(Photograph a, Photograph b)` in CompareByCaption
- `compare(Photograph a, Photograph b)` in CompareByRating

You are encouraged to keep your existing tests from HW2 as well as write tests for the other methods, but we will not require it. You will need to submit these tests along with the rest of your code. Use standard naming conventions for these JUnit tests (include the word 'test' in the beginning of the method name, such as `testRemovePhoto`). Remember to only test one thing per JUnit test case (method), also try to use only one assert statement in each JUnit test case. There is no upper limit on how many JUnit test cases you write.

Place all your JUnit test cases in one single file ("JUnit Test Case"); you do not need separate files to test each of your classes.

Style: You must follow the Coding Style Guide. This includes:

- Correct naming conventions, including appropriate camelCasing and TitleCasing.
- Comment each file, with a block at the top of the file denoting assignment information and comments for each field and method of your classes. You should also comment portions of your code that may be difficult to follow. We would like you to get into the habit of commenting your code. This adds to the readability of your code which contributes to "good quality" code.
- Use correct indentation. Eclipse makes this easy: select all code and choose "Correct Indentation" or Control-I (Windows/Linux) or Command-I (Mac).
- Do not put your classes into a package. (If you don't know what this means, don't worry about it.)
- If two methods share identical logic, you should factor that out into a separate method (a "helper" method).