Sprint 2
Albums: Keeping Photos Organized

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

- Develop a Java class that uses the Collections framework.
- Utilize `hashCode()` to determine object uniqueness.
- Define a date object, compare dates, and manipulate dates in Java.
- 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 (including the test code within your JUnit tests)

Instructions: Now that you've harnessed the power of Instagram, posting `Photograph`s to your `photos`, it's time for more power! Our photo feeds have gotten out of control and we'd like to be able to organize our photos a little better, so we've introduced Albums! To implement this additional functionality, you will need to modify your original `PhotoLibrary` and `Photograph` classes and add an `Album` class. You must adhere to the following guidelines when creating and modifying your classes:

`Photograph` class: You will need to modify your `Photograph` class from Sprint 1 by adding the following fields and methods:

- Fields:
  - `dateTaken` (private)
    A String containing the date the photograph was taken. Dates are given in the format "**YYYY-MM-DD**" such as "2019-02-13" (February 13, 2019)[1].

[1] An incorrectly formatted date may, for example, be missing one section, be in the wrong order, include the 15th month or the 42nd day. This is not an exhaustive list of examples; you should verify that the date appears as "YYYY-MM-DD".

  - `rating` (private)
    An int; the rating of the photograph on a scale from 0 to 5. No other values are allowed.

- Constructors: Provide one additional constructor that has the following header:
  `public Photograph(String filename, String caption, String dateTaken, int rating)`
  This will provide an additional way to create `Photograph` objects that will save coding when all of the necessary data are on hand. You should keep both constructors for this class.

- Accessors (AKA getters): Provide public methods that return references to each of the new fields. You must use the standard naming convention for these.

- Mutators (AKA setters): Add mutators for the `rating` and `caption` fields.

- Other Methods:

  - `public int equals(Object o)`
    Update the `equals` method to also compare the `dateTaken` value, along with `filename` and `caption`. `Photograph`s with the same caption and filename but taken on different dates should not return true.
  - `public int hashCode()`
    Override the default `hashCode` method in the `Object` class to produce a unique integer for a Photograph object. This method should return the `hashCode()` of the following String, which includes both the filename and the caption:
    `String uniqueStr = filename + "---" + caption + "---" + dateTaken;`

`PhotoLibrary` class: You will need to modify your `PhotoLibrary` class from homework 1 by adding the following fields and methods:

- Fields:

  - `albums` (private)
    A `HashSet` of `Albums` that this user has created. Each album will then contain photos from this user's `photos` stream that they have organized into albums.

- Accessors (AKA getters): Provide a new public method that returns a reference to the `albums` field. You must use the standard naming convention for accessors.

- Mutators (AKA setters): Since albums are controlled within the `PhotoLibrary` class, do not write a setter for the `albums` field. It will be modified by new or updated methods below.

- Other Methods:

  - `public ArrayList<Photograph> getPhotos(int rating)`
    Return an `ArrayList` of photos from the `photos` feed 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`.
  - `public ArrayList<Photograph> getPhotosInYear(int year)`
    Return an `ArrayList` of photos from the `photos` feed 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`.
  - `public ArrayList<Photograph> getPhotosInMonth(int month, int year)`
    Return an `ArrayList` of photos from the `photos` feed 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`.

- `public ArrayList<Photograph> getPhotosBetween(String beginDate, String endDate)`
  Return an `ArrayList` of photos from the `photos` feed 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, or `beginDate` is after `endDate`, return `null`. If there are no photos taken during the period, return an empty `ArrayList`.
- `public boolean createAlbum(String albumName)`
  Creates a new Album with name `albumName` and adds it to the list of albums, only if an Album with that name does not already exist. Returns `true` if the add was successful, `false` otherwise.
- `public boolean removeAlbum(String albumName)`
  Removes the Album with name `albumName` if an Album with that name exists in the set of albums. Returns `true` if the remove was successful, `false` otherwise.
- `public boolean addPhotoToAlbum(Photograph p, String albumName)`
  Add the Photograph `p` to the Album in the set of albums that has name `albumName` if and only if it is in the `PhotoLibrary`'s list of `photos` and it was not already in that album. Return `true` if the Photograph was added; return `false` if it was not added.
- `public boolean removePhotoFromAlbum(Photograph p, String albumName)`
  Remove the Photograph `p` from the Album in the set of albums that has name `albumName`. Return `true` if the photo was successfully removed. Otherwise return `false`.
- `private Album getAlbumByName(String albumName)`
  This is a private helper method. Given an album name, return the `Album` with that name from the set of albums. If an album with that name is not found, return `null`.
- `public boolean erasePhoto(Photograph p)`
  Modify your `erasePhoto` from homework 2 to remove the Photograph `p` from the PhotoLibrary list of photos as well as remove the Photograph from any `Albums` in the list of albums. Return `true` if the photograph was successfully removed, `false` otherwise.
- `public String toString()`
  Modify your original `toString()` method to also show a list of `Album` names contained in the album list. Any reasonable implementation of this is acceptable.

`Album` class: Albums contain a list of photos.

- Fields:
  - `name` (private)
    A String containing the Album's name in whatever form it was provided.
  - `photos` (private)
    An `ArrayList<Photograph>` of photos in the album. You are required to use `ArrayList<Photograph>`, not an array or other kind of set.
- Constructors: Provide one constructor that takes a name for the `Album`. Make sure you follow good standard Java practice and initialize all your fields in the constructor.
- Accessors (AKA getters): Provide public methods that return references to the name and photos fields. You must use the standard naming convention for these.
- Mutators (AKA setters): Write a setter for the name field but not for the photos field (which will be changed by other methods outlined below). You must use the standard naming convention for the mutator.

- Other Methods:
  - `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 album, 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 album (in `photos`).
  - `public boolean equals(Object o)`
    Following the standard rules and conventions as shown in class, return `true` if the current Album object's name value is equal to the name value of the Album object passed to `equals()`. Otherwise, return `false`.
  - `public String toString()`
    A means to print out an Album object. Generate a String that has the name of the album on the first line, followed by a list of the contained photos' filenames.
  - `public int hashCode()`
    Override the default `hashCode` method in the `Object` class produce a unique integer for an Album. This method should return the `hashCode()` of the `name` field.

Reminder: For each of the methods, you must write the exact method signature as specified. In order for us to write test cases to check your code on Web-CAT, we need to ensure everybody is using the same names for fields and methods. For example, our test cases on Web-CAT would look specifically that you have 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 using the same names we ask you to use, paying attention to capitalization, and do not change the method return type, number of type 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 in the PhotoLibrary class:

- `getPhotos(int rating)`
- `getPhotosInMonth(int month, int year)`
- `getPhotosBetween(String beginDate, String endDate)`
- `erasePhoto(Photograph p)`
- `similarity(PhotoLibrary p)` from Sprint 1

You are encouraged to write tests for the other methods but we will not require it. You will need to submit these 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 `testSimilarity`). Remember to only test one thing per JUnit test case (method). 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 `Photograph.java`, `PhotoLibrary.java`, and `Album.java`.

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).