Sprint 1
Even Instagram started somewhere...

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

- Implement code that utilizes String and primitive variable types.
- Create a Java object (class) with getters and setters.
- Develop Java classes that use instances of another class.
- Implement an ArrayList to store a collection of items.
- Implement and utilize static methods appropriately.
- Test written code using main method testing.

Grading Rubric

- 70% Methods function properly
- 30% Design and readability (See Coding Style Guide)
  - 5% Correct indentation
  - 5% Naming Conventions
  - 10% Well-commented (including the test code within your `main()` method)
  - 10% Testing in `main()`

Instructions: Even Instagram started somewhere. This is the first in a series of assignments that will produce an application for storing and viewing photographs. We will refer to each assignment as a *sprint*, a term used by developers to indicate a fixed period of time in which deliverables are produced. (Learn more about the Agile process here: https://www.agilealliance.org/agile101/.) Each sprint will produce a complete product that builds on the sprint that came before it.

In this first sprint you will be breaking ground on the photography application, beginning with the foundational classes `PhotoLibrary` and `Photograph`. These classes will define what it means to be a `Photograph`, what it means to be a `PhotoLibrary`, and how they interact; specifically a `PhotoLibrary` may contain `Photograph` items a user posts to their photo feed. You must adhere to the following requirements (guidelines) when creating your classes:

`Photograph` class: For this assignment we will consider a Photograph to be just a caption and filename.

- Fields:
  - `caption` (private)
    A String; the caption of the photograph. Once created this will never change, so you are welcome to make it `final` if you want.
  - `filename` (private)
    A String; the filename of the photograph. Once created this will never change, so you are welcome to make it `final` if you want.
- Constructors: Provide one constructor that takes a `filename` and `caption` (in that order).
- Accessors (AKA getters): Provide public methods that return references to the caption and filename fields. You must use the standard naming convention for these.
- Mutators (AKA setters): None. Once a photograph exists, its caption and filename are fixed.
- Other Methods:
  - `public boolean equals(Object o)`
    Following the standard rules and conventions as shown in class, return `true` if the Photograph object passed to equals() with caption and filename strings match (are equal to) the caption and filename strings of the current Photograph object; otherwise, return `false`.
  - `public String toString()`
    A means to print out a Photograph object. Generate a String that shows the values of the fields caption and filename. Any reasonable implementation of this is acceptable.

`PhotoLibrary` class: A PhotoLibrary has a name and a list of photos the user has posted to their photo feed, but both of those can change. To keep libraries straight, they also have a numerical ID.

- Fields:
  - `name` (private)
    A String containing the PhotoLibrary's name in whatever form it was provided.
  - `id` (private)
    An int containing the PhotoLibrary's unique id. Once set this will never change, so you are welcome to make it `final` if you want.
  - `photos` (private)
    An `ArrayList<Photograph>` of photos the user has posted to their feed in this library. You are required to use `ArrayList<Photograph>`, not an array or other kind of list.
- Constructors: Provide one constructor that takes a name and an id in that order. Make sure you follow good standard Java practice and initialize all your fields in the constructor.
- Accessors (AKA getters): Provide public methods that return the value of the id field and 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 id (which cannot change) and 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` feed 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.
  - `public boolean hasPhoto(Photograph p)`
    Return `true` if the current object has `p` in its list of `photos`. Otherwise return `false`.
  - `public boolean erasePhoto(Photograph p)`
    Once a photo is available online, it's hard to delete. However, this particular PhotoLibrary may delete the photo from its feed. If Photograph p is in the current PhotoLibrary object's list of Photographs, remove p from the current object's list. Return `true` if the Photograph was removed or `false` if it was not found.
  - `public int numPhotographs()`
    Return the number of Photographs the current object has taken (in `photos`).
  - `public boolean equals(Object o)`
    Following the standard rules and conventions as shown in class, which are in keeping with the Java equals method specification, return `true` if the current PhotoLibrary object's id value is equal to the id value of the PhotoLibrary object passed to equals(). Otherwise, return `false`.
  - `public String toString()`
    A means to print out a PhotoLibrary object. Generate a String representation of a PhotoLibrary object showing the values of the name, id, and photos fields. Any reasonable implementation of this is acceptable.
  - `public static ArrayList<Photograph> commonPhotos(PhotoLibrary a, PhotoLibrary b)`
    Return an `ArrayList<Photograph>` of the photos that both PhotoLibrary a and PhotoLibrary b have posted to their feeds. Use the equals method of the `Photograph` class to determine if two Photograph objects represent the same photograph.
  - `public static double similarity(PhotoLibrary a, PhotoLibrary b)`
    Returns a measure of how similar the photo feeds are between PhotoLibrary a and PhotoLibrary b, in terms of a numerical value between 0 and 1. If either PhotoLibrary does not have any photos, the result is 0.0. Otherwise, it is the number of `commonPhotos` to both libraries divided by smaller of the number of photos in a's feed and the number of photos in b's feed.
    Reminder: Java respects types, so the integer division of `3 / 4` gives integer `0`, while including a float or double in the division `3 / 4.0` gives `0.75`.

NOTE: For each of the methods, you must write the exact method signature as specified.

Testing: We expect that you will add a `main()` method to test your code. We require at least **two** tests of each of the methods listed above, excluding the getters and setters. Part of the Design and Readability grade is based on of your main method testing.

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).