Sprint 4+
Graphical User Interface

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

- Experience writing code using Event-Driven Programming.
- Design and develop a Graphical User Interface using Swing.
- Implement try/catch blocks to handle Exceptions.
- Develop features described by requirements rather than detailed specifications.

Grading Rubric

- 55% Functionality
- 15% User interface
  - 5% User interface is intuitive
  - 5% User interface is aesthetically pleasing
  - 5% Overall interface structure is logical and natural
- 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 main-method code within your `main()` method

In this final sprint, you will be utilizing all the of the previously implemented logic in a graphical user interface. You will not have a client to make interface design decisions for you, which means that some creative license is required. You will need to make some assumptions about how this software will best work. **Throughout this assignment, comment your code with any assumptions you make.** There are intentionally features in this assignment beyond the in-class material – it

is expected you will use the Java documentation to look up APIs to allow you to execute these requirements. **Cite sources in the code comments.**

Instructions: The project owner wants a Graphical User Interface (GUI) front-end to the `PhotoLibrary` application that we've been building. To start development of this application, you have been assigned to develop a prototype of the album viewer GUI using Swing.

Photograph: You should modify your `Photograph` class to add the image data of the photograph.

- Fields:
  - `imageData` (**protected**)
    A `BufferedImage` object that contains the image's data.
- Methods:
  - `public boolean loadImageData(String filename)`
    Given a filename as a String, this method loads the Image data from the file and stores it into the `imageData` field. The `filename` is stored in the `filename` field. This method returns `true` on success and `false` on failure.
    An Image can be loaded from a String using the following code:
    `BufferedImage img = ImageIO.read(new File(filename));`
- Accessors (AKA getters): Provide a public method that returns a reference to the new field. You must use the standard naming convention for it.
- Mutators (AKA setters): Add a mutator for the new field.

PhotoViewer: Add a new class, `PhotoViewer`, to define and launch the GUI interface.

- `PhotoViewer` should have an instance variable, `imageLibrary`, that is a `PhotographContainer`.
  e.g. `private PhotographContainer imageLibrary;`
- Provide getters and setters for the `imageLibrary` field.
- Any form or GUI components needed to implement the requirements should also be created as instance variables.
- Create a folder named `images` in the root of your Eclipse project and put 5 images in it. At least some of these images should be 1280 x 1024 or larger – typical sizes for photos taken by a modern camera. By locating your images folder here, Java will be able to find the images using the relative path below.
- In the main method of the `PhotoViewer` class, write a test invocation that creates a `PhotographContainer` containing 5 hard-coded images and loads your prototype GUI, like the example in Figure 1.
- The window will require four main areas:
  - An area for displaying the current image
  - An area for a thumbnail images - one thumbnail image for each of the five photos in the album
  - An area for form controls
  - An area for rating the current image

Layout of these spaces is up to you, the developer, but most of the window should display the current image, which should default to the first image in the collection. An example layout is depicted in Figure 2.

- Thumbnail-sized images (having a width of 200 pixels or less) should display down the left or right side of the window or across the top or bottom of the window, one thumbnail for each image in the collection.
- In addition to the image, thumbnails should provide a caption that displays the image caption, as well as the date and rating.
- Clicking on a thumbnail image should cause that image to be displayed in the current image area.
- **Note** Using a `JLabel` component will allow you to add a thumbnail image as an `ImageIcon`. The `JLabel` component will also allow you to set a caption to display caption, date and rating information, as well as add a `MouseListener` object. View the Java docs on how to write a MouseListener: https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html.
- By default, order the thumbnails by date.
- Provide a control by which the thumbnails can be ordered by Date, Caption or Rating. Selecting a new sort order should update the list of displayed thumbnail images and reset the current image to the first image in the album.
- Provide **Previous** and **Next** buttons that, when clicked, advance the image in the main display to the previous or next image in the container, respectively.
- When the end of the container is reached, clicking the **Next** button should display the first image.
- When the first image is selected, clicking the **Previous** button should display the last image.
- Provide a rating feature with radio buttons labeled 1 through 5, respectively. When an image is displayed in the current image area, the corresponding rating radio button should be selected matching the rating for that `Photograph`. If another rating radio button is selected, the rating value for that `Photograph` should be updated and reflected in the list of thumbnail images.

Hint 1: The appearance of GUI applications are dependent on screen resolution. Laptops come in a wide variety of screen resolutions, and it is likely your grader may not have the same resolution. For this reason, it is wise to avoid fixed-size windows. Take advantage of the built-in Swing features (Layouts) to size your overall GUI window. Size images consistently to keep the image presentation uniform and avoid skewing or stretching the image. Include a screenshot with your submission.

Hint 2: You will need to use multiple panels and multiple layouts to achieve the desired display.

Full documentation and examples of Swing components are available at `https://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html`.

Style: 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).

```
public static void main(String[] args) {

    PhotoViewer myViewer = new PhotoViewer();

    // the relative image directory (use one line from below)
    String imageDirectory = "images/";  // for Macs/Linux, or
    String imageDirectory = "images\\"; // for Windows

    Photograph p1 =  new Photograph("Caption", imageDirectory +
       "img1.jpg", "2015-06-30", 5);
    // add four more photographs like the line above

    myViewer.setImageLibrary(new PhotoLibrary("Test Library", 1));

    myViewer.getImageLibrary().addPhoto(p1);
    // four more photographs added like the line above

    // use Collections.sort() to sort the photos based on the
    // written compareTo() method
    Collections.sort(myViewer.getImageLibrary().getPhotos());

    // Invoke and start the GUI
    javax.swing.SwingUtilities.invokeLater(()->myViewer.createAndShowGUI());
    // or equivalently using an anonymous class
    javax.swing.SwingUtilities.invokeLater(new Runnable {
        public void run() {
            // This method is run to create and show the GUI
            myViewer.createAndShowGUI();
        }
   });
}
```

Figure 1: An example main method to load images and invoke the GUI.

Figure 2: Example layout of PhotoViewer GUI. You do not need to match this layout exactly. Feel free to be creative, as long as you meet all of the requirements.