# JLITEBOX: An Image Viewer

# Dexter Legaspi

Table of Contents

[JLITEBOX: An Image Viewer 1](#_Toc96261495)

[Dexter Legaspi 1](#_Toc96261496)

[Assignment 1 4](#_Toc96261497)

[1.1 SUMMARY DESCRIPTION EVALUATION CRITERION (i) APPLIES 5](#_Toc96261498)

[1.2 I/O EXAMPLE FROM *PROJECTED* COMPLETED PROJECT *EVALUATION CRITERION (i) APPLIES* 5](#_Toc96261499)

[1.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE *EVALUATION CRITERION (ii) APPLIES* 6](#_Toc96261500)

[1.3.1 Import Images 7](#_Toc96261501)

[1.3.1 Save Imported Images to Final Destination folder 7](#_Toc96261502)

[1.3.2 Identify JPEG vs RAW Images 7](#_Toc96261503)

[1.4 ILLUSTRATIVE OUTPUT FROM IMPLEMENTATION *EVALUATION CRITERION (ii) APPLIES* 7](#_Toc96261504)

[Provide illustrative output from your implemented application (so far) showing that the requirements have been met. Explain what class.method(s) produce it. 7](#_Toc96261505)

[1.5 YOUR DIRECT*OR*Y 9](#_Toc96261506)

[Show a screenshot of your directory. This should include a parallel directory of JUnit tests where possible—package-by-package, class-by-class, and method-by-method, except for trivial ones. 9](#_Toc96261507)

[1.6 TECHNIQUES IMPLEMENTED *EVALUATION CRITERION (iii) APPLIES* 10](#_Toc96261508)

[1.6.1 Class model and Sequence Diagram 10](#_Toc96261509)

[1.6.2 Code showing an abstract class or interface 11](#_Toc96261510)

[1.6.3 Code showing polymorphism 12](#_Toc96261511)

[1.6.4 Code showing upcasting or downcasting 12](#_Toc96261512)

[1.7 YOUR CODE 13](#_Toc96261513)

[1.8 EVALUATION OF ASSIGNMENT 1 13](#_Toc96261514)

[Assignment 2 13](#_Toc96261515)

[2.1 SUMMARY DESCRIPTION, UPDATED AS APPLICABLE 13](#_Toc96261516)

[*EVALUATION CRITERION (i) APPLIES* 13](#_Toc96261517)

[2.2 I/O EXAMPLE FROM PROJECTED COMPLETED PROJECT, UPDATED AS APPLICABLE *EVALUATION CRITERION (i) APPLIES* 14](#_Toc96261518)

[2.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE NOT IMPLEMENTED BEFORE *EVALUATION CRITERION (i) APPLIES* 14](#_Toc96261519)

[2.3.1 Open a File of List of URLs for Import 14](#_Toc96261520)

[2.3.2 Download Files from Remote Location 14](#_Toc96261521)

[2.3.3 Read Image Metadata 15](#_Toc96261522)

[The application reads metadata embedded in the image file and persists it in memory for logging and/or display in the user interface. The number of metadata tags are displayed in the logs. 15](#_Toc96261523)

[2.4 I/O EVIDENCE THAT THE ABOVE FUNCTIONALITY WAS ACHIEVED 15](#_Toc96261524)

[*EVALUATION CRITERION (ii) APPLIES* 15](#_Toc96261525)

[This typically consists of screen shots of input and output, together with text explaining their context. 15](#_Toc96261526)

[2.5 YOUR DIRECT*OR*Y 16](#_Toc96261527)

[2.6 TECHNIQUES IMPLEMENTED *EVALUATION CRITERION (iii) APPLIES* 17](#_Toc96261528)

[2.6.1 Class model and Sequence Diagram 17](#_Toc96261529)

[2.6.2 Code showing *file I/O* 18](#_Toc96261530)

[2.6.3 Code showing *exception*, preferably a user-defined exception 18](#_Toc96261531)

[2.7 YOUR CODE 19](#_Toc96261532)

[2.8 EVALUATION OF ASSIGNMENT 2 20](#_Toc96261533)

[Appendix 1 (will be read as-needed only—add more as necessary) 20](#_Toc96261534)

[Assignment 3 20](#_Toc96261535)

[3.1 SUMMARY DESCRIPTION, UPDATED AS NECESSARY 20](#_Toc96261536)

[*EVALUATION CRITERION (i) APPLIES* 20](#_Toc96261537)

[3.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE 21](#_Toc96261538)

[*EVALUATION CRITERION (i) APPLIES* 21](#_Toc96261539)

[3.2.1 Filter Catalog List with User-Specified Criteria 21](#_Toc96261540)

[3.2.2 Render the RAW Image to the User Interface 21](#_Toc96261541)

[3.2.3 Identify the Equipment Used to Capture Image 21](#_Toc96261542)

[3.3 I/O EVIDENCE OF ACCOMPLISHING THE REQUIREMENTS LISTED ABOVE 21](#_Toc96261543)

[*EVALUATION CRITERION (ii) APPLIES* 21](#_Toc96261544)

[Log Output for Filtering: 22](#_Toc96261545)

[GUI Output for Rendering RAW Image and Metadata: 22](#_Toc96261546)

[3.4 YOUR DIRECT*OR*Y 23](#_Toc96261547)

[3.5 YOUR UPDATED CLASS MODEL AND CLARIFICATION OF HOW THE EXECUTION WORKS 24](#_Toc96261548)

[*EVALUATION CRITERION (i) APPLIES* 24](#_Toc96261549)

[3.6 WHERE GENERICS ARE IMPLEMENTED 25](#_Toc96261550)

[*EVALUATION CRITERION (iii) APPLIES* 25](#_Toc96261551)

[3.6.1 Class model fragment showing generic class 25](#_Toc96261552)

[3.6.2 Code (including test code), input (if applicable), and output showing generics 29](#_Toc96261553)

[3.7 YOUR CODE 29](#_Toc96261554)

[*EVALUATION CRITERION (iii) APPLIES* 29](#_Toc96261555)

[3.8 INSTRUCTOR’S EVALUATION 30](#_Toc96261556)

[Assignment 4 31](#_Toc96261557)

[4.1 SUMMARY DESCRIPTION 31](#_Toc96261558)

[4.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE 31](#_Toc96261559)

[4.2.1 Persist Imported Image Metadata into a Permanent Storage 32](#_Toc96261560)

[4.2.2 Retrieve Image Metadata of Images from Catalog from a Permanent Storage 32](#_Toc96261561)

[4.2.3 Persist Image Preview of Imported Images into a Permanent Storage 32](#_Toc96261562)

[4.2.4 Retrieve Preview of Imported Images from a Permanent Storage 32](#_Toc96261563)

[4.2.5 Render Preview of Imported Images into a Permanent Storage 32](#_Toc96261564)

[4.2.6 Import Images from the GUI 32](#_Toc96261565)

[4.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE 32](#_Toc96261566)

[Metadata and Preview Persistence 33](#_Toc96261567)

[Input / Output 33](#_Toc96261568)

[4.4 YOUR DIRECTORY 36](#_Toc96261569)

[4.5 DESIGN 37](#_Toc96261570)

[4.5.1 Use Case: Saving Metadata and Preview to Permanent Storage 37](#_Toc96261571)

[4.6 CODE SNIPPETS 39](#_Toc96261572)

[4.6.1 Code showing object read and write 39](#_Toc96261573)

[4.6.2 Code Showing *stream*() and Lambdas 42](#_Toc96261574)

[4.7 YOUR CODE 43](#_Toc96261575)

[4.8 YOUR CODE 43](#_Toc96261576)

[4.9 Evaluation 44](#_Toc96261577)

[Assignment 5 44](#_Toc96261578)

[5.1 SUMMARY DESCRIPTION 44](#_Toc96261579)

[5.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE 45](#_Toc96261580)

[5.2.1 Display ALL the Images from the Catalog in a Grid 45](#_Toc96261581)

[5.2.3 Optionally Hide Metadata in Grid 45](#_Toc96261582)

[5.2.4 Optionally Show a Convenient Toolbar 45](#_Toc96261583)

[5.2.5 Optionally Show RAW Metadata on Image Mouse Hover 45](#_Toc96261584)

[5.2.5 Optionally Resize the Image Preview 46](#_Toc96261585)

[5.2.6 Import Images from the Background without Blocking UI 46](#_Toc96261586)

[5.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE 47](#_Toc96261587)

[Input/Output 47](#_Toc96261588)

[5.4 YOUR DIRECTORY 52](#_Toc96261589)

[5.5 DESIGN 54](#_Toc96261590)

[5.6 CODE SNIPPETS 56](#_Toc96261591)

[5.6.1 Code showing where concurrency is *defined* 56](#_Toc96261592)

[} 57](#_Toc96261593)

[5.6.2 Code showing where concurrency is *used* 58](#_Toc96261594)

[5.7 YOUR CODE 58](#_Toc96261595)

[5.8 Evaluation 58](#_Toc96261596)

[Assignment 6 2/16/2021 12:04 PM 59](#_Toc96261597)

[6.1 SUMMARY DESCRIPTION 59](#_Toc96261598)

[6.2 KEY REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE 60](#_Toc96261599)

[6.2.1 Navigate Individual Image in the Catalog with a Larger Preview 60](#_Toc96261600)

[6.2.2 Filter the Grid with Images that Meet Specified Filter 60](#_Toc96261601)

[6.2.3 Open an Image with Default App 60](#_Toc96261602)

[6.2.4 Show Catalog Statistics 61](#_Toc96261603)

[6.2.4 Export Catalog as JSON 61](#_Toc96261604)

[6.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE 62](#_Toc96261605)

[Input/Output 62](#_Toc96261606)

[6.4 YOUR DIRECTORY 63](#_Toc96261607)

[6.5 DESIGN 64](#_Toc96261608)

[6.5.1 Class Model, Use Case, and Sequence Diagram 64](#_Toc96261609)

[Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red where you applied the features listed below. 64](#_Toc96261610)

[6.6 CODE SNIPPETS 66](#_Toc96261611)

[6.6.1 Code of any kind showing table creation 66](#_Toc96261612)

[6.6.2 Java code involving the insertion of data 67](#_Toc96261613)

[6.6.3 Java code involving the selection of records, and ordering with SQL 69](#_Toc96261614)

[6.6.4 Java code involving selection involving at least two tables 69](#_Toc96261615)

[6.6.5 Java code involving selection and aggregation of data 69](#_Toc96261616)

[6.7 YOUR CODE 70](#_Toc96261617)

[6.9 Evaluation 70](#_Toc96261618)

# Assignment 1

For this assignment, you will describe and implement release 1 of your term project. You will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*. You are free to choose a project that interests you but if you prefer, the instructor and your facilitator will be happy to suggest a topic. If you are already an experienced developer, this is an opportunity to build a challenging application (check with your facilitator if it requires significant API’s) or discuss research with the instructor. It is OK to name a project with much more scope than you can accomplish in the course (as in the example above): we will not hold you to completing everything associated with it. What we do expect is that you specify and implement a set of do-able requirements within such scope.

Submit this completed Word document. Replace as indicated. Please observe and retain the gray text. Your materials—in black 12-point Times New Roman—should not exceed 5 pages excluding references, figures, and appendices. Use the Appendix sections for additional material if you need to. These will be read only on an as-needed basis.

We want you to develop in Eclipse preferably or else IntelliJ (talk to your facilitator about exceptions). As you code, use JUnit tests whenever possible but certainly by week 2—package-by-package, class-by-class, and method-by-method, except for trivial methods and those requiring I/O. Use non-Junit classes for testing the latter. Keep the evaluation criteria in mind, listed at the end.

For this assignment, you do not need to read data from a file—you can build all data into the code.

Include a ReadMe file describing where to run the application from, and including necessary execution notes. All JUnit tests will be assumed runnable.

# 1.1 SUMMARY DESCRIPTION *EVALUATION CRITERION (i) APPLIES*

One- or two-paragraph overall description of your proposed term project—half-page (12-point Times New Roman) limit. By the end, term projects will incorporate most of the techniques discussed in the course. To do this, you may need to alter the direction of your project or introduce an additional project in future. You may alter this or even replace it as the semester progresses. You will probably find it useful to use your project acronym.

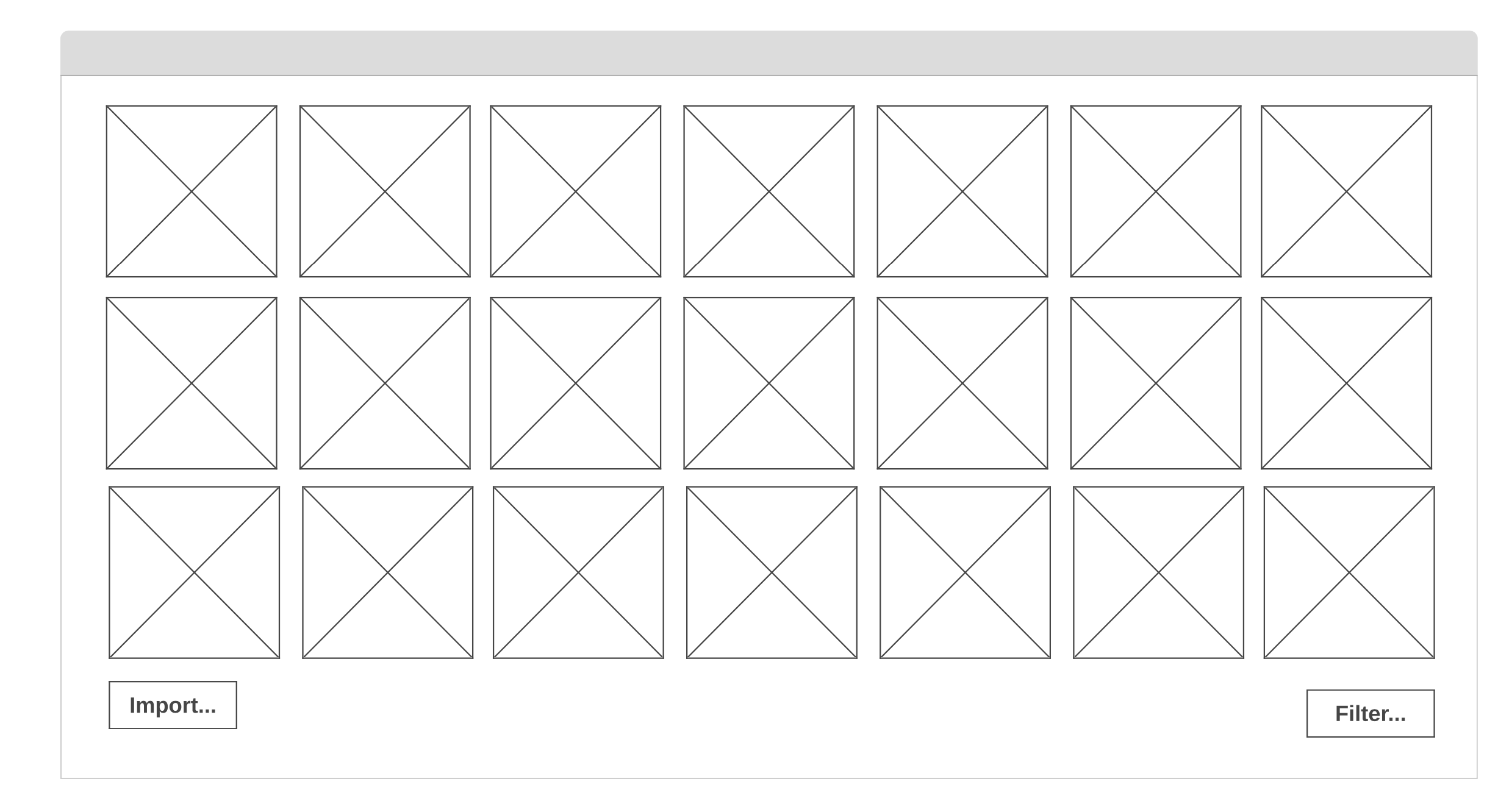
JLiteBox is an application to import digital camera images (JPG and various RAW formats; see [this explanation](https://helpx.adobe.com/lightroom-cc/how-to/raw-vs-jpeg.html) for the difference between the formats.) to store them in a common repository and be able to view them with simple and convenient navigation scheme akin to a virtual image lightbox. This is basically a scaled down version of [Adobe Lightroom](https://www.adobe.com/products/photoshop-lightroom.html) or [Camera Bits PhotoMechanic](https://home.camerabits.com/) but with just the browsing and tagging capabilities (i.e., no image editing capabilities)

It will be leveraging the [JavaFX](https://openjfx.io/index.html) libraries to create the GUI and be able to render images on the screen and navigate through them individually and also show them in a grid format and use existing open-source libraries to extract image metadata (e.g., camera/lens information). There will be a means to filter images and only show a subset of the images from the catalog in the grid. The pertinent metadata will be stored in a local database for speed and efficiency.

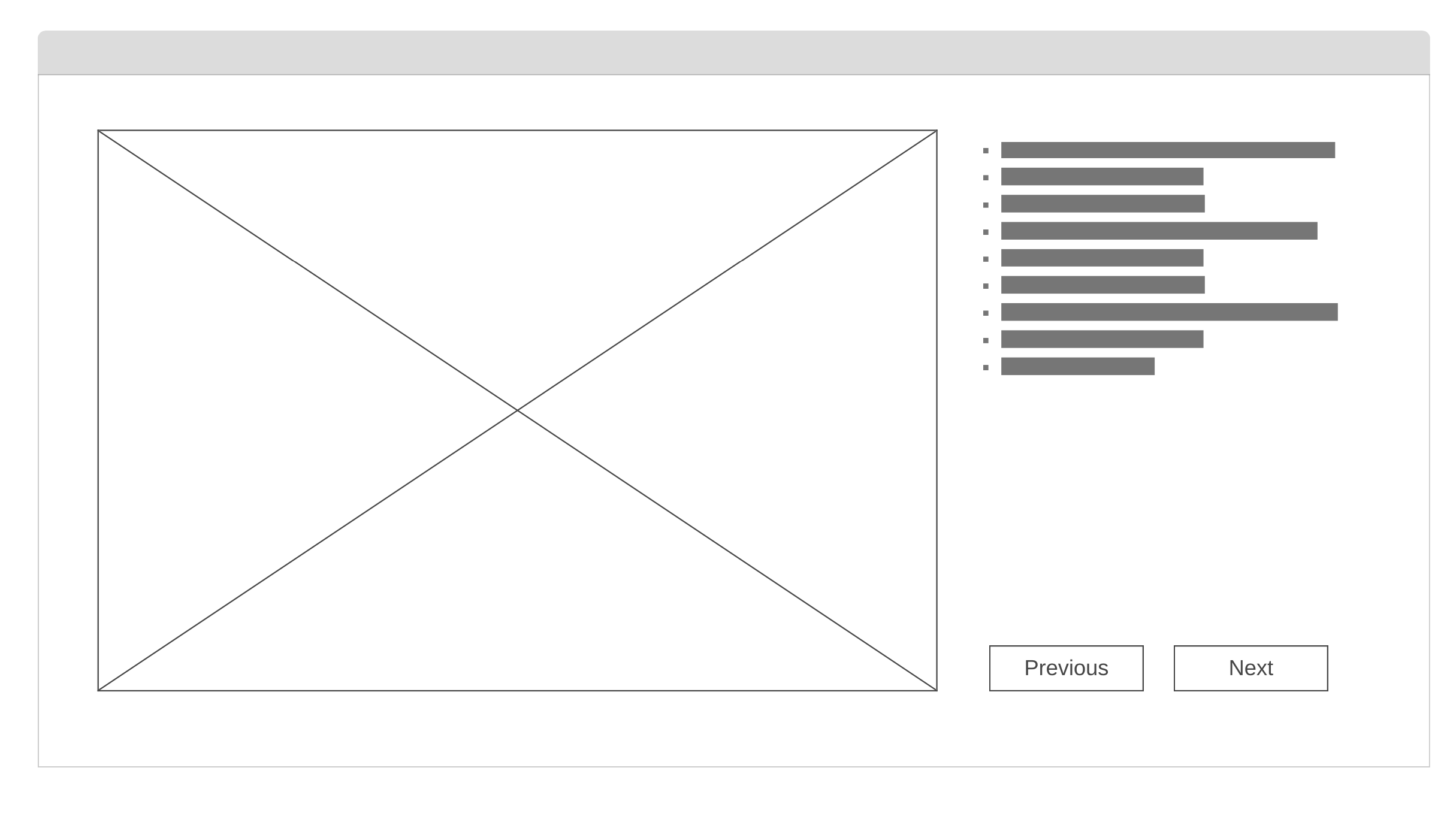
## 1.2 I/O EXAMPLE FROM *PROJECTED* COMPLETED PROJECT *EVALUATION CRITERION (i) APPLIES*

Provide an example of projected *concrete* output for designated input. You will not be held to fulfilling exactly this—it is just explanatory at this point, to indicate where your project is going. We recognize that project direction and details will change as the term progress. This section refers to the project as a whole, not just to what you will produce this week, so we can gain an idea of what you have in mind overall.

The user is presented with a GUI for the catalog. It will have the option to show all the images (with filtering options as shown in the wireframe below).



It will also have the option to cycle through the images of the catalog individually to have a larger view of the Image along with the metadata.



Lastly, it will have the option to import new images into the catalog when the click the “Import…” button as shown in the first wireframe. User will be presented with a dialog box of the source of the images. Once user clicks “OK”, the images will be copied into the catalog.

Activity of the catalog will be written to console or a log file depending on how the logging is configured.

## 1.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE *EVALUATION CRITERION (ii) APPLIES*

Supply [functional requirements](https://docs.google.com/document/d/1eU7eINLDxmrf793D4OF2yGT4ry_SW3GQGoVDYzecGHc/edit?usp=sharing) statement that you accomplished for this assignment, i.e., functionality that the application provides for the user. Please state requirement in declarative form, as illustrated in the examples, because here we want to know the functionality intended (*what*, not *how*). For example, the following is *not* a proper functional requirement: *TicTac will have a class for O’s and a class for X’s.* It is common to mistake design elements like this for requirements. To get started, state what the application will accept as input, like requirement 1.3.1 below.

Keep in mind that the implementation of your requirements will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*; that will probably influence the requirements you choose to implement in this assignment. The example material supplied should be deleted before you submit.

### 1.3.1 Import Images

JLiteBox will import the images from the specified source directory reference the image file at a later time and be able to render the images on the UI.

### 1.3.1 Save Imported Images to Final Destination folder

Imported images are copied to the final destination folder to be able to rehydrate the Image on the next time the application is opened.

### 1.3.2 Identify JPEG vs RAW Images

JLiteBox will be able to identify the JPEG images vs RAW images and will log them accordingly. It will also report that if the image is RAW it will generate a Preview image (this is only logged, no image preview is generated yet).

## 1.4 ILLUSTRATIVE OUTPUT FROM IMPLEMENTATION *EVALUATION CRITERION (ii) APPLIES*

### Provide illustrative output from your implemented application (so far) showing that the requirements have been met. Explain what class.method(s) produce it.

The app does not have a GUI yet, it will only accept a src directory from a command line argument

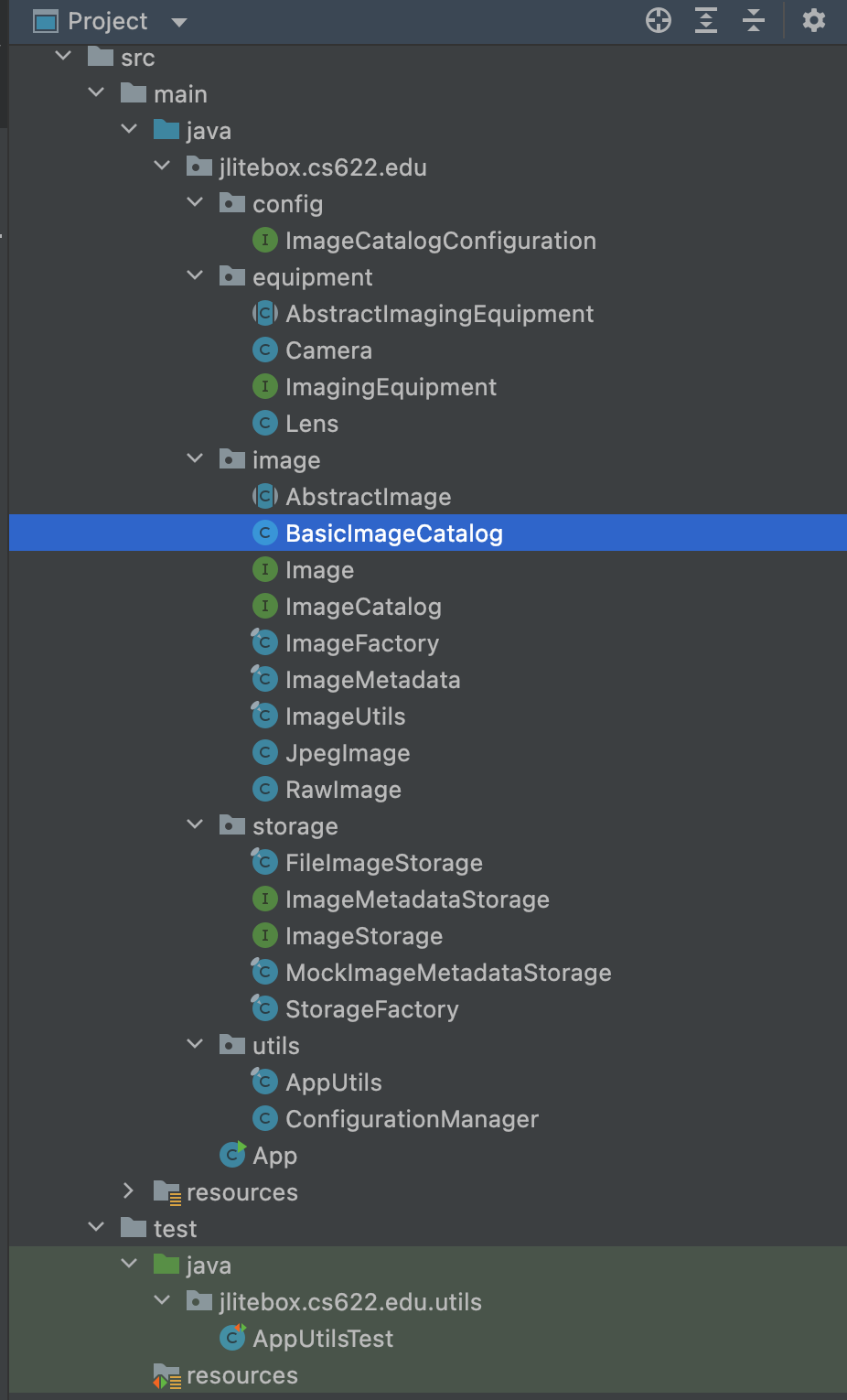
The following is produced by logging feature of the application that logs to the screen, the files from the directory passed as a command line argument:

|  |
| --- |
| 21:51:39.019 [main] INFO jlitebox.cs622.edu.App - JLiteBox 1.0-SNAPSHOT.7fdece7  21:51:39.030 [main] INFO j.cs622.edu.image.BasicImageCatalog - Adding images from dir /Users/dexter/Projects/CS622/bu-ms-s5-01/sample/images  21:51:39.052 [main] INFO j.cs622.edu.image.BasicImageCatalog - R1002 is a JPG Image  21:51:39.053 [main] INFO j.cs622.edu.image.BasicImageCatalog - R1001 is a JPG Image  21:51:39.053 [main] INFO j.cs622.edu.image.BasicImageCatalog - DSC\_2304 is a RAW image of type NEF  21:51:39.053 [main] INFO j.cs622.edu.image.BasicImageCatalog - DSC\_2305 is a RAW image of type NEF  21:51:39.053 [main] INFO j.cs622.edu.image.BasicImageCatalog - R0000978 is a RAW image of type DNG  21:51:39.053 [main] INFO j.cs622.edu.image.BasicImageCatalog - R0000979 is a RAW image of type DNG  21:51:39.054 [main] INFO jlitebox.cs622.edu.image.RawImage - Generating Image Preview for DSC\_2304  21:51:39.055 [main] INFO jlitebox.cs622.edu.image.RawImage - Generating Image Preview for DSC\_2305  21:51:39.055 [main] INFO jlitebox.cs622.edu.image.RawImage - Generating Image Preview for R0000978  21:51:39.055 [main] INFO jlitebox.cs622.edu.image.RawImage - Generating Image Preview for R0000979  21:51:39.055 [main] INFO j.cs622.edu.storage.FileImageStorage - Destination directory is /var/folders/19/nbx8wtx57f3b1cbtqtbgqg1w0000gn/T/  21:51:39.055 [main] INFO j.cs622.edu.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-01/sample/images/R1002.JPG  21:51:39.064 [main] INFO j.cs622.edu.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-01/sample/images/R1001.JPG  21:51:39.070 [main] INFO j.cs622.edu.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-01/sample/images/DSC\_2304.NEF  21:51:39.113 [main] INFO j.cs622.edu.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-01/sample/images/DSC\_2305.NEF  21:51:39.155 [main] INFO j.cs622.edu.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-01/sample/images/R0000978.DNG  21:51:39.204 [main] INFO j.cs622.edu.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-01/sample/images/R0000979.DNG  21:51:39.253 [main] INFO j.c.e.s.MockImageMetadataStorage - Saving metadata for R1002  21:51:39.253 [main] INFO j.c.e.s.MockImageMetadataStorage - Saving metadata for R1001  21:51:39.253 [main] INFO j.c.e.s.MockImageMetadataStorage - Saving metadata for DSC\_2304  21:51:39.253 [main] INFO j.c.e.s.MockImageMetadataStorage - Saving metadata for DSC\_2305  21:51:39.253 [main] INFO j.c.e.s.MockImageMetadataStorage - Saving metadata for R0000978  21:51:39.254 [main] INFO j.c.e.s.MockImageMetadataStorage - Saving metadata for R0000979  21:51:39.254 [main] INFO j.cs622.edu.image.BasicImageCatalog - Catalog has 6 images |

## 1.5 YOUR DIRECTORY

### Show a screenshot of your directory. This should include a parallel directory of JUnit tests where possible—package-by-package, class-by-class, and method-by-method, except for trivial ones.

The class hierarchy are defined for the Image types as well as the ImagingEquipment types although the classes/interfaces for the ImagingEquipment are largely not used yet.



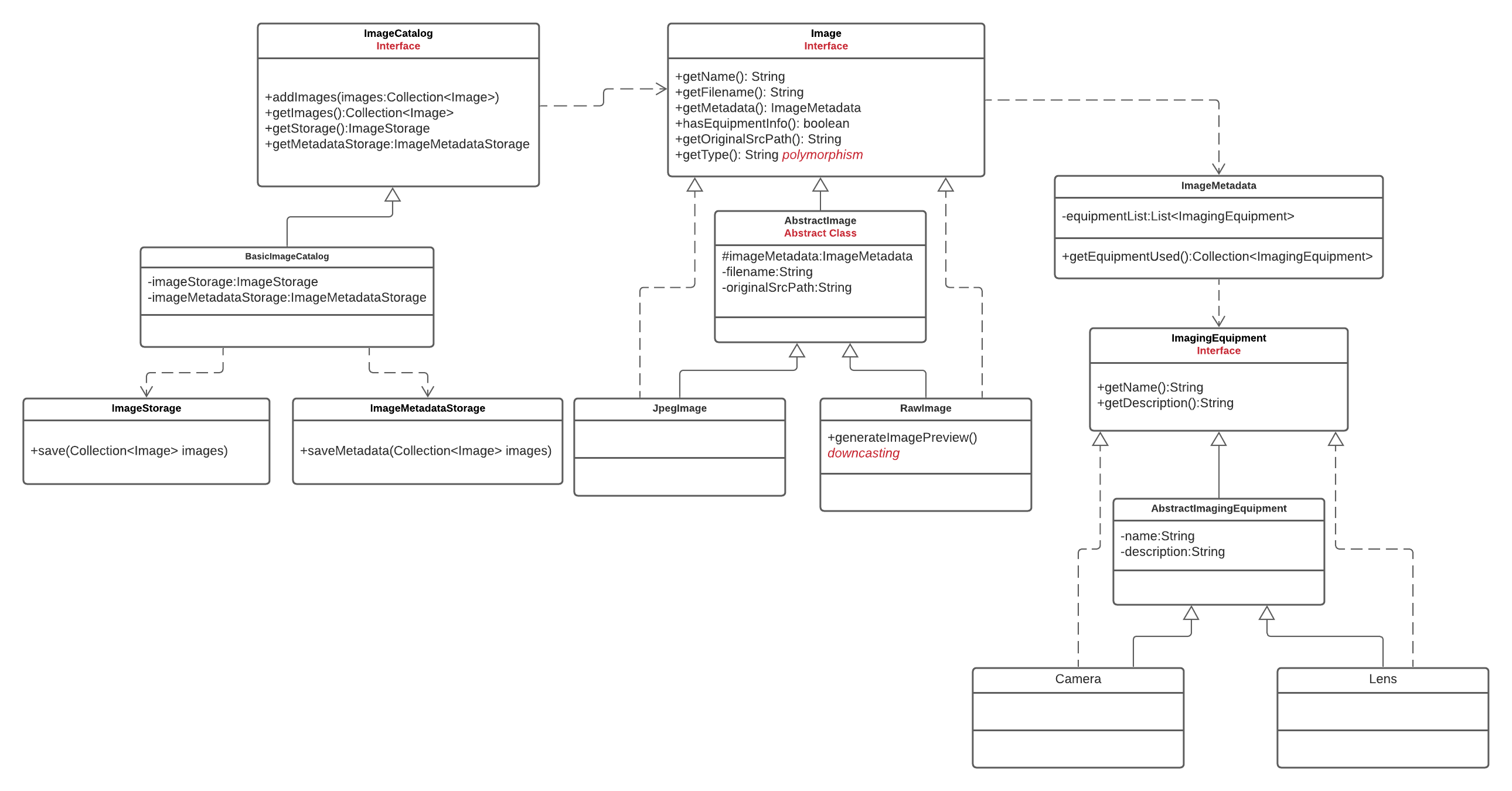
## 1.6 TECHNIQUES IMPLEMENTED *EVALUATION CRITERION (iii) APPLIES*

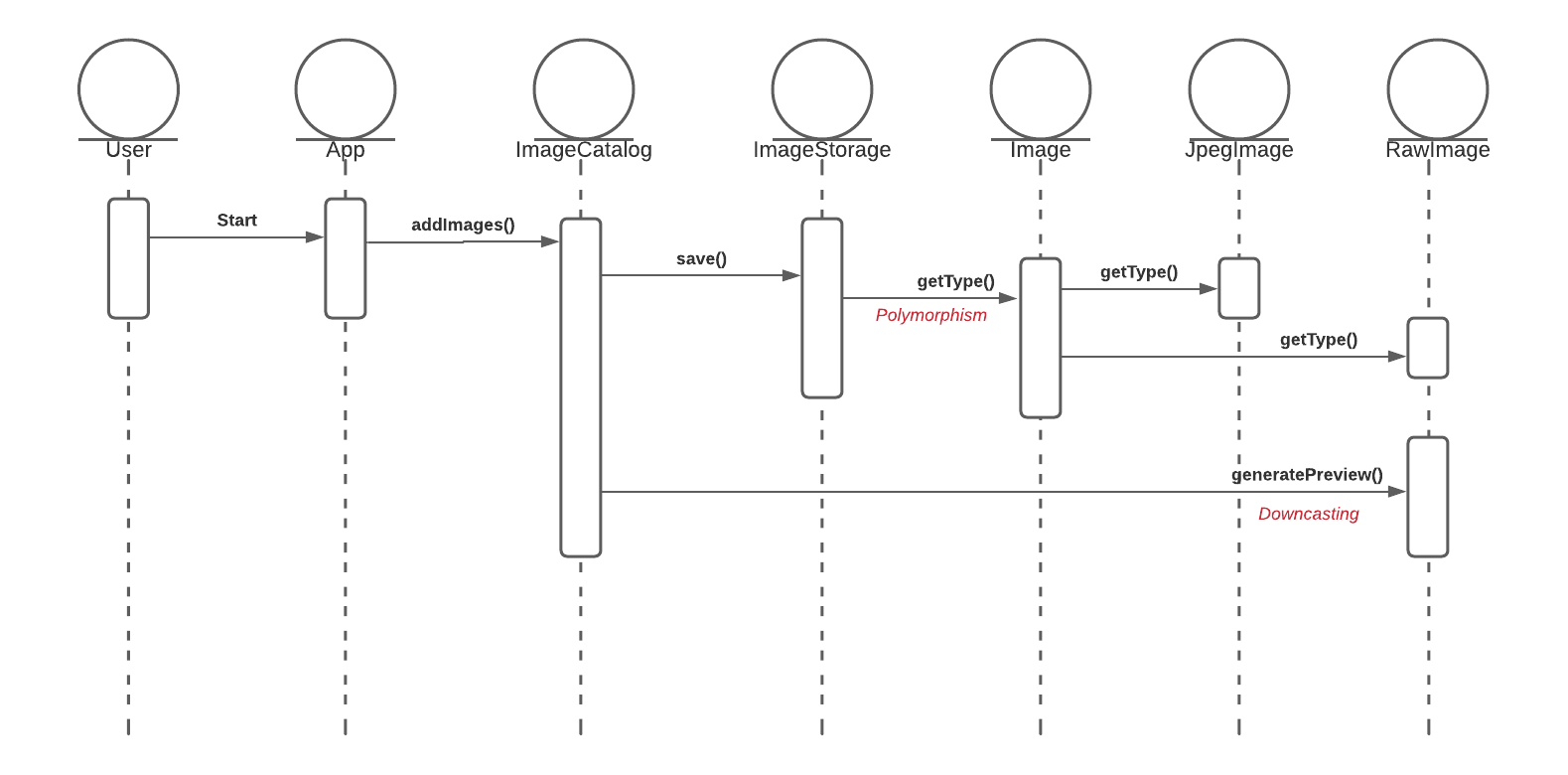
Your implementation should include *inheritance*, *polymorphism*, and *either an abstract class or interface* at least once, and in a manner that is useful to your application. Explain where and how you applied these, using the headings below.

### 1.6.1 Class model and Sequence Diagram

Identify where you included *inheritance*, *polymorphism*, and *abstract classes* or *interfaces* in your class model. Make classes and members *static* or not as per their intended usage. To do this use tools (e.g., Visio and Lucidchart), PowerPoint, or a combine models as in [this example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to cut and paste from). Insert indications in red (as in [this example](https://docs.google.com/spreadsheets/d/1ZvkerE9FkWHWwVGdzuy7YMBU6oBMFGZbA4sotFETs8Y/edit?usp=sharing)) to show where the three features below apply.

There are two main hierarchies in the application: first is based on the Image interface to represent the two classes or images (JPEG and RAW) and second is based on the ImagingEquipment interface to represent the equipment used to capture the images (camera, lens, etc).





There are other interfaces and corresponding class implementations to support (design is currently in flux):

1. ImageCatalog and BasicImageCatalog represents the class that contains the Image objects
2. ImageStorage and FileImageStorage represents the filesystem where the Image files will be stored
3. ImageMetadataStorage represents the (database) for the metadata of the images.

### 1.6.2 Code showing an abstract class or interface

Show the relevant code (only) and explain why an abstract class or interface is appropriate here. It should be clear where the code is located (class and method).

Image is an abstract class that lists the base behavior and methods of an image that would have the common properties/methods that applies to all types of images that is supported in the catalog.

|  |
| --- |
| package jlitebox.cs622.edu.image;  import jlitebox.cs622.edu.utils.AppUtils;  public abstract class Image {  protected ImageMetadata metadata;  private String filename;  private String originalSrcPath;  public String getFilename() {  return filename;  }  @Override  public ImageMetadata getMetadata() {  return metadata;  }  public String getOriginalSrcPath() {  return originalSrcPath;  }  Image(@NonNull String originalSrcPath) {  this.filename = AppUtils.getFilename(originalSrcPath);  this.originalSrcPath = originalSrcPath;  }  public String getName() {  return AppUtils.getBaseFilename(getFilename());  }  public boolean hasEquipmentInfo() {  return getMetadata().getEquipmentUsed() != null &&  getMetadata().getEquipmentUsed().size() > 0;  }  } |

### 1.6.3 Code showing polymorphism

Show the relevant code (only) and explain why *polymorphism* is appropriate here. Recall that polymorphism is implemented in one of two ways – overriding methods in subclasses or overloading methods in the same class where the method signatures are different – and allowing the language runtime to dynamically invoke the correct method. It should be clear where the code is located (class and method).

The method BasicImageCatalog#addImages shows a simple demonstration of polymorphism; the getType() returns a String that describes its (image) type:

|  |
| --- |
| // list the types  newImages.forEach(i -> {  logger.info("{} is a {}", i.getName(), i.getType());  }); |

### 1.6.4 Code showing upcasting or downcasting

Show the relevant code (only) and explain why upcasting or downcasting is appropriate here. It should be clear where the code is located (class and method).

The method BasicImageCatalog#addImages shows a simple demonstration of downcasting. Since RAW images are the ones that require to generate image previews, they are filtered out accordingly here:

|  |
| --- |
| // only RAW images need to generate JPG image previews  newImages.stream()  .filter(i -> i instanceof RawImage)  .forEach(i -> ((RawImage) i).generateImagePreview(imageStorage)); |

## 1.7 YOUR CODE

--1/20/2021 9:07 AM

## 1.8 EVALUATION OF ASSIGNMENT 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Section(s)** | **D** | **C** | **B** | **A** | **Letter Grade** | **%** |
| **(i) 1.1 Summary Description and 1.2 I/O Examples and 1.3 Requirements** | Hard to understand. | Understandable | … *and* concise *and* all requirements declarative statements of functionality | … *and* extremely clear *and* complete functionality can be built from the requirement statements alone. | A- | 90.0 |
| **(ii) 1.4 Illustrative output from application** | Minimal requirements met. | Some committed requirements are clearly met | All committed requirements are met *and* they are non-trivial | … in clear correspondence with all requirement details. | A- | 90.0 |
| **(iii) 1.6 Techniques implemented** | Design or code hard to understand | Design understandable, *and* code mostly understandable from comments | … *and* concise (no redundant parts) | … *and* all methods precisely specified and very clear. | B+ | 87.0 |
|  |  |  |  | Assignment Grade: |  | 89.0 |

# Assignment 2

Implement the next release of your term project. You will incorporate exception handling and file I/O, as specified below.

Submit this completed Word document, including your name within the file name, observing and retaining the gray text like this. Retain the headings or edit them as indicated. Your Assignment 2 materials—in black 12-point Times New Roman—should not exceed 5 pages excluding references, figures, and appendices. Use the Appendix sections for additional material if you need to. These will be read on an as-needed basis.

As you code, use JUnit tests—package-by-package, class-by-class, and method-by-method, except for trivial ones and ones requiring I/O that cannot be unit tested. Use non-Junit classes for testing the latter, as in the worked example.

Include a ReadMe file describing where to run the application from, and including notes as necessary (not more).

Keep the evaluation criteria in mind, listed at the end and referred to in the headings.

## 2.1 SUMMARY DESCRIPTION, UPDATED AS APPLICABLE

## *EVALUATION CRITERION (i) APPLIES*

One- or two-paragraph overall description of your whole proposed term project. Edit your last description as needed.

JLiteBox is an application to import digital camera images (JPG and various RAW formats; see [this explanation](https://helpx.adobe.com/lightroom-cc/how-to/raw-vs-jpeg.html) for the difference between the formats.) to store them in a common repository and be able to view them with simple and convenient navigation scheme akin to a virtual image lightbox. This is basically a scaled down version of [Adobe Lightroom](https://www.adobe.com/products/photoshop-lightroom.html) or [Camera Bits PhotoMechanic](https://home.camerabits.com/) but with just the browsing and tagging capabilities (i.e., no image editing capabilities)

It will be leveraging the [JavaFX](https://openjfx.io/index.html) libraries to create the GUI and be able to render images on the screen and navigate through them individually and also show them in a grid format and use existing open-source libraries to extract image metadata (e.g., camera/lens information). There will be a means to filter images and only show a subset of the images from the catalog in the grid. The pertinent metadata will be stored in a local database for speed and efficiency.

There are also two options to import files without having to launch the GUI: one is using by specifying the directory and the other is specifying a file with a list of URLs where the application can download them and add to the catalog. It emits log messages as the image are downloaded and since network connections are more fragile than local directory, the application offers a way to recover if one file (to skip it) has an issue with downloading.

## 2.2 I/O EXAMPLE FROM PROJECTED COMPLETED PROJECT, UPDATED AS APPLICABLE *EVALUATION CRITERION (i) APPLIES*

Provide an example of projected *concrete* output for example input, indicating how users will interact with your application. You will not be held to fulfilling exactly this—it intended to help us understand the probable direction of your application as a whole.

In addition to what was specified in [1.2](#_1.2_I/O_EXAMPLE), when the user opted to import images from a file, there is an interactive, command-line only version that allows a user interaction in case a file download fails and asks if the user wants to skip that URL and continue or abort the entire import altogether:

|  |
| --- |
| 17:44:49.990 [main] INFO edu.bu.cs622.jlitebox.App - JLiteBox 1.0-SNAPSHOT.8f36f99  17:44:50.455 [main] INFO edu.bu.cs622.jlitebox.App - Importing from file sample/import.txt  17:44:50.456 [main] INFO e.b.c.j.image.BasicImageCatalog - there are 2 urls in file sample/import.txt  17:44:50.458 [main] INFO e.b.c.j.image.BasicImageDownloader - Saving https://live.staticflickr.com/65535/51836490494\_3fa5339ee8\_k.jpg to /var/folders/19/nbx8wtx57f3b1cbtqtbgqg1w0000gn/T/images/51836490494\_3fa5339ee8\_k.jpg  Error processing https://live.staticflickr.com/65535/51836490494\_3fa5339ee8\_k.jpg Continue? [y, n]? |

## 2.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE NOT IMPLEMENTED BEFORE *EVALUATION CRITERION (i) APPLIES*

### 2.3.1 Open a File of List of URLs for Import

The application optionally allows the user to specify a path to a file as a command line argument has an option to read a list of URLs from a file path specified as a command line argument which is read internally in a list for downloading.

### 2.3.2 Download Files from Remote Location

The application allows user to specify a command line parameter that specifies a path of a text file that has the list of URLs. From the list of URLs, the application can download images from a remote location and save it to the image catalog. The application makes a network connection and download the file to a local temporary directory for further processing.

### 2.3.3 Read Image Metadata

### The application reads metadata embedded in the image file and persists it in memory for logging and/or display in the user interface. The number of metadata tags are displayed in the logs.

## 2.4 I/O EVIDENCE THAT THE ABOVE FUNCTIONALITY WAS ACHIEVED

## *EVALUATION CRITERION (ii) APPLIES*

## This typically consists of screen shots of input and output, together with text explaining their context.

The application opens an optional file and downloads the images from the individual URLs and copies them to the final destination in the catalog

|  |
| --- |
| 17:31:35.892 [main] INFO edu.bu.cs622.jlitebox.App - JLiteBox 1.0-SNAPSHOT.8f36f99  17:31:36.337 [main] INFO edu.bu.cs622.jlitebox.App - Importing from file sample/import.txt  17:31:36.339 [main] INFO e.b.c.j.image.BasicImageCatalog - there are 2 urls in file sample/import.txt  17:31:36.340 [main] INFO e.b.c.j.image.BasicImageDownloader - Saving https://live.staticflickr.com/65535/51836490494\_3fa5339ee8\_k.jpg to /var/folders/19/nbx8wtx57f3b1cbtqtbgqg1w0000gn/T/images/51836490494\_3fa5339ee8\_k.jpg  17:31:36.802 [main] INFO e.b.c.j.image.BasicImageDownloader - Saving https://live.staticflickr.com/65535/51815575646\_3ec1d92c83\_k.jpg to /var/folders/19/nbx8wtx57f3b1cbtqtbgqg1w0000gn/T/images/51815575646\_3ec1d92c83\_k.jpg  17:31:36.840 [main] INFO e.b.c.j.image.BasicImageCatalog - 51836490494\_3fa5339ee8\_k is a JPG Image  17:31:36.841 [main] INFO e.b.c.j.image.BasicImageCatalog - 51815575646\_3ec1d92c83\_k is a JPG Image  17:31:36.841 [main] INFO e.b.c.j.storage.FileImageStorage - Destination directory is /tmp  17:31:36.841 [main] INFO e.b.c.j.storage.FileImageStorage - Copying image /var/folders/19/nbx8wtx57f3b1cbtqtbgqg1w0000gn/T/images/51836490494\_3fa5339ee8\_k.jpg  17:31:36.846 [main] INFO e.b.c.j.storage.FileImageStorage - Copying image /var/folders/19/nbx8wtx57f3b1cbtqtbgqg1w0000gn/T/images/51815575646\_3ec1d92c83\_k.jpg  17:31:37.004 [main] INFO e.b.c.j.i.ExifToolMetadataExtractor - Metadata successfully extracted from 51836490494\_3fa5339ee8\_k; it has 72 metadata tags  17:31:37.142 [main] INFO e.b.c.j.i.ExifToolMetadataExtractor - Metadata successfully extracted from 51815575646\_3ec1d92c83\_k; it has 69 metadata tags  17:31:37.143 [main] INFO e.b.c.j.s.MockImageMetadataStorage - Saving metadata for 51836490494\_3fa5339ee8\_k  17:31:37.143 [main] INFO e.b.c.j.s.MockImageMetadataStorage - Saving metadata for 51815575646\_3ec1d92c83\_k  17:31:37.143 [main] INFO e.b.c.j.image.BasicImageCatalog - Catalog has 2 images |

## 2.5 YOUR DIRECTORY

Show a screenshot of your directory. This should include a parallel directory of JUnit tests—package-by-package, class-by-class, and method-by-method, except for trivial ones.

The JUnit tests for some of the classes are combined as it would be more exhaustive to test them in combination rather than individually. JaCoCo mocking framework was also leveraged to be able to mock some of the interfaces where applicable to have a more predictable unit test results.

|  |  |
| --- | --- |
| Text  Description automatically generated | Text  Description automatically generated |

## 2.6 TECHNIQUES IMPLEMENTED *EVALUATION CRITERION (iii) APPLIES*

Your implementation should exploit *file IO* and *exceptions* at least once, in as natural a manner as possible. Using the headings below, explain where and how you applied these.

### 2.6.1 Class model and Sequence Diagram

Indicate clearly in your class model where you applied file IO and exception handling, including a user-defined exception if possible. “Enforce what you intend.” For example, make classes and members *static* or not as per their intended usage. To do this use tools, PowerPoint, or combine models as in [this RUML example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to copy, cut and paste from). Insert indications in red (as in the example) to show where the three features below apply.

The updated class hierarchy showing the Exception classes

Diagram

Description automatically generated

The link for the class diagram can be found [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

Here is the Sequence Diagram showing the Exception handling. This is loosely patterned from [here](https://johanvergeer.github.io/posts/uml-sequence-diagram-exception-handling).

Diagram, schematic

Description automatically generated

The link for the sequence diagram can be found [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-479%2C-12%2C1659%2C1075%2CYqgNN~vfhlMh&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

### 2.6.2 Code showing *file I/O*

Show the relevant code (only). It should be clear where the code is located (class and method). Specify nontrivial methods with pre- and postconditions (and examples if this clarifies).

BasicImageCatalog::addImagesFromFile shows a File being read into a list and downloads all the image files from the URL list and saves them to a temporary directory. Any Exception like IOException is wrapped in a custom JLiteBoxException so it can be handled accordingly in the controller class.

|  |
| --- |
| @Override  public int addImagesFromFile(String path,  ImageImportOptions options, Predicate<String> onErrorCallback)  throws JLiteBoxException {  List<Image> newImages = new ArrayList<>();  int addedImages = 0;  try {  var urls = Files.readAllLines(Paths.get(path));  logger.info("there are {} urls in file {}", urls.size(), path);  for (var url : urls) {  try {  // pull the image from the url and save it to the temp directory  var tmpFilePath = downloader.download(url);  newImages.add(ImageFactory.fromFile(tmpFilePath));  } catch (Exception e) {  if (!onErrorCallback.test(url))  throw e;  }  }  addedImages = addImages(newImages, options);  downloader.cleanup();  } catch (Exception e) {  if (!options.isIgnoreErrors())  throw new JLiteBoxException("Error in adding images from " + path, e);  }  return addedImages;  } |

### 2.6.3 Code showing *exception*, preferably a user-defined exception

Show the relevant code (only) and explain why *exceptions* are appropriate here. It should be clear where the code is located (class and method). Specify nontrivial methods with pre- and postconditions (and examples if this clarifies).

From the same code shown in [2.6.2](#_2.6.2_Code_showing), with network connection there can be a few things that could happen that’s related to network connectivity, transmission and saving them to a temporary file and they should be handled but allows recoverability; with errorCallback, the view can get an input from the user to continue if one file fails and skip to the next or abort the entire import altogether.

|  |
| --- |
| try {  // pull the image from the url and save it to the temp directory  var tmpFilePath = downloader.download(url);  newImages.add(ImageFactory.fromFile(tmpFilePath));  } catch (Exception e) {  if (!onErrorCallback.test(url))  throw e;  } |

|  |
| --- |
| @Override  public String download(String url) throws JLiteBoxException {  var fileName = AppUtils.getFilename(url);  var fullTmpDestPath = AppUtils.createPath(workingDir, fileName);  try {  // pull the image from the url and save it to the temp directory  logger.info("Saving {} to {}", url, fullTmpDestPath);  var u = new URL(url);  var conn = u.openConnection();  var is = new BufferedInputStream(conn.getInputStream());  var file = new File(fullTmpDestPath);  FileUtils.copyInputStreamToFile(is, file);  return fullTmpDestPath;  } catch (Exception e) {  throw new JLiteBoxException*(*e.getMessage*()*, e*)*;  }  } |

## 2.7 YOUR CODE

Unless your facilitator requests or allows another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request an exception. Specify nontrivial methods with pre- and postconditions (and examples if this clarifies). For excellent work, specify the class invariants.

The code is in a separate zip file.

## 2.8 EVALUATION OF ASSIGNMENT 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Section(s)** | **D** | **C** | **B** | **A** | **Letter Grade** | **%** |
| **(i) 2.1 Summary Description and 2.2 I/O Examples and 2.3 Requirements** | Hard to understand. | Understandable | … and concise and all requirements are declarative statements of functionality | … and extremely clear and complete functionality can be built from the requirement statements alone. | B | 85.0 |
| **(ii) 2.4 I/O evidence that the above functionality was achieved** | Minimal requirements met. | Some committed requirements are clearly met and a ReadMe file is included. | All committed requirements are met and they are non-trivial | … and in clear correspondence with all requirement details. | B+ | 87.0 |
| **(iii) 2.6 Techniques implemented and 2.7 Your code** | Design or code hard to understand | Design understandable, and code mostly understandable from comments | … and concise (no redundant parts) and appropriately applied | … and all methods precisely specified and very clear. | B- | 80.0 |
|  |  |  |  | Assignment Grade: |  | 84.0 |

## Appendix 1 (will be read as-needed only—add more as necessary)

1/28/2021 7:03 AM

# Assignment 3

Implement the next release of your term project (preferably, or start a new one if you have to). You will incorporate generics. The same instructions as in Assignment 2 apply to this completed Word document, the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

## 3.1 SUMMARY DESCRIPTION, UPDATED AS NECESSARY

*EVALUATION CRITERION (i) APPLIES*

One- or two-paragraph overall description of your proposed term project. Color in red the parts different from Assignment 2.

JLiteBox is an application to import digital camera images (JPG and various RAW formats; see [this explanation](https://helpx.adobe.com/lightroom-cc/how-to/raw-vs-jpeg.html) for the difference between the formats.) to store them in a common repository and be able to view them with simple and convenient navigation scheme akin to a virtual image lightbox. This is basically a scaled down version of [Adobe Lightroom](https://www.adobe.com/products/photoshop-lightroom.html) or [Camera Bits PhotoMechanic](https://home.camerabits.com/) but with just the browsing and tagging capabilities (i.e., no image editing capabilities)

It will be leveraging the [JavaFX](https://openjfx.io/index.html) libraries to create the GUI and be able to render images on the screen and navigate through them individually and also show them in a grid format and use existing open-source libraries to extract image metadata (e.g., camera/lens information). There will be a means to filter images and only show a subset of the images from the catalog in the grid. The pertinent metadata will be stored in a local database for speed and efficiency.

There are also two options to import files without having to launch the GUI: one is using by specifying the directory and the other is specifying a file with a list of URLs where the application can download them and add to the catalog. It emits log messages as the image are downloaded and since network connections are more fragile than local directory, the application offers a way to recover if one file (to skip it) has an issue with downloading.

## 3.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

*EVALUATION CRITERION (i) APPLIES*

Title and one or two sentences per requirement. Don’t repeat requirements implemented for prior assignments unless they are necessary to provide context—in which case, make it clear which are new vs. old.

### 3.2.1 Filter Catalog List with User-Specified Criteria

The application allows to select a subset of images in the catalog to be returned (mostly for display purposes) so that the user can list/display the images that he/she wants based on some filter criteria specified.

### 3.2.2 Render the RAW Image to the User Interface

The application will create a preview and render image of a RAW image file on the graphical user interface depending on the contents of the catalog.

### 3.2.3 Identify the Equipment Used to Capture Image

The application will identify the equipment used to capture a RAW/JPEG image based on the embedded metadata in the file. It will identify the make/brand and model if they exist otherwise mark the camera and/or lens as “Unknown.”

## 3.3 I/O EVIDENCE OF ACCOMPLISHING THE REQUIREMENTS LISTED ABOVE

*EVALUATION CRITERION (ii) APPLIES*

Provide an example of actual input / output corresponding to the requirements above

The filter is not yet wired to any user input yet, so for now it’s an arbitrary filter based on the name that starts with ‘R’. Similarly, the rendering of the Image is not tied to the Catalog itself nor to any Controllers, so for now there is a hard-coded file location that renders the corresponding

### Log Output for Filtering:

Console I/O:

Text

Description automatically generated

### GUI Output for Rendering RAW Image and Metadata:

The GUI is showing a RAW Image rendered as JavaFx Image along with the Camera/Lens and other metadata (aperture, shutter speed)

Graphical user interface, text, application, chat or text message

Description automatically generated

## 3.4 YOUR DIRECTORY

Show a screenshot of your directory. This should include a parallel directory of JUnit tests where applicable—package-by-package, class-by-class, and method-by-method, except for trivial and inapplicable ones.

Source Directory Structure:

|  |  |
| --- | --- |
| A picture containing table  Description automatically generated | A picture containing graphical user interface  Description automatically generated |

Unit Tests Directory Structure:

|  |
| --- |
| Graphical user interface  Description automatically generated with medium confidence |

## 3.5 YOUR UPDATED CLASS MODEL AND CLARIFICATION OF HOW THE EXECUTION WORKS

*EVALUATION CRITERION (i) APPLIES*

Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent and clear. Indicate clearly in your class model where you applied generics. To do this use tools, PowerPoint, or a combine models as in [this example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to cut and paste from). Insert indications in red to show where generics apply.

The ImageCatalog uses FilteredImagesCollection to filter the image it would need to render based on the criteria specified by the ImageContentFilter. The FilteredImagesCollection is a subclass of generic FilteredContentCollection. This is a generic because the container does not need to know what it contains; it just needs to know it can filter (using the ImageContentFilter). Similarly, the ImageContentFilter is also a generic because it does not need to know where it is used; it only needs to “test” if a criterion (or criteria) is met. The driving need for this is due to user interaction where the view (along with the controller) will have to create a “filter object” based on what the user wants to filter on. And with the use of generics, the catalog will be abstracted on how the filter will be applied.

The use of generics here makes it reusable to other object type T other than Image. In the unit test FilteredContentTests#testBasicGenerics the generic is used for other types.

Diagram

Description automatically generated

Larger version can be viewed [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-178%2C-515%2C1659%2C1075%2C0_0&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

Diagram

Description automatically generated

Larger version can be viewed [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-138%2C-3%2C1659%2C1075%2CXlQO5MwRXDsg&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

## 3.6 WHERE GENERICS ARE IMPLEMENTED

*EVALUATION CRITERION (iii) APPLIES*

### 3.6.1 Class model fragment showing generic class

Explain where and how you applied *generic classes* in your class model.

There is a generic class definition FilteredContentCollection<T, F extends ContentFilter<T>> which defines a container that can filter content based on ContentFilter<T>. The FilteredContentCollection does not know nor care about the implementation of the filter ContentFilter<T>; it just knows that filter will operate on type T and that it can “test” if a criteria/criterion of the filter is met—i.e., the FilteredContentCollection does not even need to know about the filter criteria! The generic also allows it to be flexible and reusable by not having to be bounded.

|  |
| --- |
| public class FilteredContentCollection<T, F extends ContentFilter<T>>  implements Iterable<T> {  private final List<T> internalList;  public FilteredContentCollection(Collection<T> contents) {  // make the list thread-safe so we are using CopyOnWriteArrayList  this.internalList = new CopyOnWriteArrayList<>(contents);  }  /\*\*  \* filter the list with the specialized filter  \*  \* @param filter the filter object  \* @return the filtered list in the collection  \*/  public Collection<T> filterWith(F filter) {  return filterWith(filter::test);  }  /\*\*  \* This is just a backdoor for filtering directly with a Predicate just added  \* for posterity  \*  \* @param filter the filter predicate  \* @return the filtered list in collection  \*/  public Collection<T> filterWith(Predicate<T> filter) {  return internalList.stream().filter(filter).collect(Collectors.toList());  }  /\*\*  \* convenience method for iteration  \*  \* @return the iterator  \*/  @Override  public Iterator<T> iterator() {  return internalList.iterator();  }  } |

|  |
| --- |
| public interface ContentFilter<T> {  /\*\*  \* Test the item if it matches filter  \*  \* @param item item to test  \* @return true if passes  \*/  boolean test(T item);  } |

|  |
| --- |
| public class FilteredImageContentCollection  extends FilteredContentCollection<Image, ImageContentFilter> {  /\*\*  \* Filtered images collection  \*  \* @param contents collection of images to filter (usually from a Catalog)  \*/  public FilteredImageContentCollection(Collection<Image> contents) {  super(contents);  }  } |

For the subclass of ContentFilter<Image>, ImageContentFilter, we are using a builder design pattern to make it flexible to create a filter as the filter criteria is not finalized yet and by using the builder design pattern, if we decide to add/remove criteria there is less susceptibility of existing code to be changed too much (and thus less prone to bugs). Note that this merely shows the skeleton of the code to show the structure of the builder class so as not to make this section overbearing; the entire class definition can be seen in the source code with the accompanying comments.

|  |
| --- |
| public final class ImageContentFilter  implements ContentFilter<Image> {  private final ImageCatalog.LogicalOperator operator;  private final Optional<String> name;  private final Optional<String> fileExt;  private final Optional<String> cameraName;  private final Optional<String> lensName;  private final Optional<Float> aperture;  private final Optional<Float> focalLength;  private final Optional<Integer> iso;  private final Optional<Float> shutterSpeed;  /\*\*  \* Builder for the ImageContentFilter  \*/  public static class ImageContentFilterBuilder {  private final boolean isAnd;  private String name;  private String fileExt;  private String cameraName;  private String lensName;  private float aperture;  private int focalLength;  private int iso;  private float shutterSpeed;  public ImageContentFilterBuilder(boolean isAndOperator) {  this.isAnd = isAndOperator;  }  // ...other with\*() methods  public ImageContentFilter build() {  return new ImageContentFilter(isAnd, name, fileExt,  cameraName, lensName, aperture, focalLength, iso,  shutterSpeed);  }  }  private ImageContentFilter(boolean isAnd, String name, String fileExt,  String cameraName, String lensName,  float aperture, float focalLength,  int iso, float shutterSpeed) {  this.operator = isAnd ?  ImageCatalog.LogicalOperator.AND : ImageCatalog.LogicalOperator.OR;  this.name = Optional.ofNullable(name);  this.fileExt = Optional.ofNullable(fileExt);  this.cameraName = Optional.ofNullable(cameraName);  this.lensName = Optional.ofNullable(lensName);  this.focalLength = focalLength == 0 ? Optional.empty() : Optional.of(focalLength);  this.iso = iso == 0 ? Optional.empty() : Optional.of(iso);  this.aperture = aperture == 0 ? Optional.empty() : Optional.of(aperture);  this.shutterSpeed = shutterSpeed == 0 ? Optional.empty() : Optional.of(shutterSpeed);  }  public boolean isAndOperation() {  return this.operator == ImageCatalog.LogicalOperator.AND;  }  @Override  public boolean test(Image image) {  // shown in the next section  }  } |

The ImageContentFilter::test definition can be seen here. As noted earlier, how the matching is done is all encompassed here and isolated (and therefore testable) and allows the class that uses it abstracted from the implementation of it.

|  |
| --- |
| @Override  public boolean test(Image image) {  var meta = image.getMetadata();  // matching of fields; we are using lambdas here to be more explicit  // what fields we are matching and to avoid repetition of code  // in the next section; also for readability  Function<CharSequence, Boolean> nameMatch =  v -> image.getName().contains(v);  Function<CharSequence, Boolean> fileExtMatch =  v -> image.getFilename().contains(v);  Function<CharSequence, Boolean> cameraMatch =  v -> meta.getCamera().map(c -> c.getBrand().contains(v)).orElse(false);  Function<CharSequence, Boolean> lensMatch =  v -> meta.getLens().map(l -> l.getBrand().contains(v)).orElse(false);  Function<Float, Boolean> focalLengthMatch =  v -> meta.getLens().map(l -> l.getFocalLength() == v).orElse(false);  Function<Float, Boolean> shutterSpeedMatch =  v -> meta.getShutterSpeed() == v;  Function<Float, Boolean> apertureMatch =  v -> meta.getAperture() == v;  Function<Integer, Boolean> isoMatch =  v -> meta.getIso() == v;  if (isAndOperation()) {  // all match  return name.map(nameMatch).orElse(true)  && fileExt.map(fileExtMatch).orElse(true)  && cameraName.map(cameraMatch).orElse(true)  && lensName.map(lensMatch).orElse(true)  && focalLength.map(focalLengthMatch).orElse(true)  && shutterSpeed.map(shutterSpeedMatch).orElse(true)  && aperture.map(apertureMatch).orElse(true)  && iso.map(isoMatch).orElse(true);  // @formatter:on  } else {  // any match  return name.map(nameMatch)  .or(() -> fileExt.map(fileExtMatch))  .or(() -> cameraName.map(cameraMatch))  .or(() -> lensName.map(lensMatch))  .or(() -> focalLength.map(focalLengthMatch))  .or(() -> shutterSpeed.map(shutterSpeedMatch))  .or(() -> aperture.map(apertureMatch))  .or(() -> iso.map(isoMatch))  .orElse(false);  }  } |

### 3.6.2 Code (including test code), input (if applicable), and output showing generics

Explain why the use of *generics* is appropriate here.

As noted already, the ImageCatalog uses the generics to filter content for UI display/render. Until the GUI is fully implemented and realized, for now the Controller simply outputs the names of the filtered images in the logs.

|  |
| --- |
| @Override  public int importImagesFromDirectory(String path, ImageCatalog.ImageImportOptions imageImportOptions)  throws JLiteBoxException {  view.emit("Importing from path: {}", path);  var imported = catalog.addImagesFromDirectory(path, imageImportOptions);  // todo: remove later  var filter = new ImageContentFilter  .ImageContentFilterBuilder(false)  .withName("R")  .build();  var filteredImages = catalog.getImages(filter);  view.emit("images starting with 'R':");  filteredImages.forEach(i -> {  view.emit(i.getName());  });  // todo: remove ends  return imported;  } |

And the log output:

|  |
| --- |
| 09:23:38.590 [main] INFO edu.bu.cs622.jlitebox.AppFx - JLiteBox 1.0-SNAPSHOT.e820398  09:23:38.598 [main] INFO edu.bu.cs622.jlitebox.AppFx - Launching app with console...  09:23:38.911 [main] INFO edu.bu.cs622.jlitebox.AppFx - Importing from directory sample/images  09:23:38.912 [main] INFO e.b.c.j.v.ImageImporterConsoleView - Importing from path: sample/images  09:23:38.912 [main] INFO e.b.c.j.image.BasicImageCatalog - Adding images from dir sample/images  ...  09:23:41.985 [main] INFO e.b.c.j.image.BasicImageCatalog - Catalog has 6 images  09:23:41.991 [main] INFO e.b.c.j.v.ImageImporterConsoleView - images starting with 'R':  09:23:41.991 [main] INFO e.b.c.j.v.ImageImporterConsoleView - R1002  09:23:41.991 [main] INFO e.b.c.j.v.ImageImporterConsoleView - R1001  09:23:41.991 [main] INFO e.b.c.j.v.ImageImporterConsoleView - R0000978  09:23:41.991 [main] INFO e.b.c.j.v.ImageImporterConsoleView - R0000979  Process finished with exit code 0 |

## 3.7 YOUR CODE

*EVALUATION CRITERION (iii) APPLIES*

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request another transmission process (e.g., github).

## 3.8 INSTRUCTOR’S EVALUATION

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criterion** | **D** | **C** | **B** | **A** | **Letter Grade** | **%** |
| **(i) 3.1 Summary Description and 3.2 Requirements and 3.5 Updated Class Model** | Hard to understand. | Understandable | … and concise and all requirements are declarative statements of functionality | … and extremely clear and complete functionality can be built from the requirement statements alone. | A | 95.0 |
| **(ii) 3.3 I/O evidence that the above functionality was achieved** | Minimal requirements met. | Some committed requirements are clearly met and a ReadMe file is included. | All committed requirements are met and they are non-trivial | … and in clear correspondence with all requirement details. | A | 95.0 |
| **(iii) 3.6 Techniques implemented and 3.7 Your code** | Design or code hard to understand | Implementation works and design is understandable, and code mostly understandable from comments | … and concise (no redundant parts) and appropriately applied and code understandable from comments | … and all methods precisely specified and very clear. | A- | 90.0 |
|  |  |  |  | Assignment Grade: |  | 93.3 |
|  |  |  |  |  |  |  |
| The resulting grade is the average of these, using A+=97, A=95, A-=90, B+=87, B=85, B-=80 etc. The maximum is 100. | | | | | |  |
| To obtain an A grade for the course, your weighted average should be >93. A-:>=90. B+:>=87. B:>83. B-:>=80 etc. | | | | | |  |

2/2/2021 4:45 PM

# Assignment 4

Implement the next release of your term project or start a new one if necessary. You will incorporate—at a minimum—the *saving and retrieval of objects*, the use of *lambdas*, and the use of *streams*. You can substitute the use of JavaFX for one of these three concepts if you wish. Also, substitutions for any of these concepts by another advanced techniques are acceptable if you have already used them: please clear this with your facilitator.

The same instructions as in Assignment 3 apply to this completed Word document regarding the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

## 4.1 SUMMARY DESCRIPTION

*Evaluation criterion (i) applies*

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from Assignment 2, if any.

JLiteBox is an application to import digital camera images (JPG and various RAW formats; see [this explanation](https://helpx.adobe.com/lightroom-cc/how-to/raw-vs-jpeg.html) for the difference between the formats.) to store them in a common repository and be able to view them with simple and convenient navigation scheme akin to a virtual image lightbox. This is basically a scaled down version of [Adobe Lightroom](https://www.adobe.com/products/photoshop-lightroom.html) or [Camera Bits PhotoMechanic](https://home.camerabits.com/) but with just the browsing and tagging capabilities (i.e., no image editing capabilities)

It will be leveraging the [JavaFX](https://openjfx.io/index.html) libraries to create the GUI and be able to render images on the screen and navigate through them individually and also show them in a grid format and use existing open-source libraries to extract image metadata (e.g., camera/lens information). There will be a means to filter images and only show a subset of the images from the catalog in the grid. The pertinent metadata will be stored in a local database for speed and efficiency. In addition, since generating a preview is an expensive process (which involves reading the original file and then generating the preview), the previews will be stored in the database along with the metadata, as reading the smaller binary stream will be much more efficient.

Since the application is essentially a read only view of the image catalog, it will provide a means in its GUI to either show the file in the filesystem (where the originals are stored) similar to “Show in Finder” in MacOS and/or a means to launch the application associated with the file (usually this will be Adobe Photoshop)

There are also two options to import files without having to launch the GUI: one is using by specifying the directory and the other is specifying a file with a list of URLs where the application can download them and add to the catalog. It emits log messages as the image are downloaded and since network connections are more fragile than local directory, the application offers a way to recover if one file (to skip it) has an issue with downloading.

## 4.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

*Evaluation criterion (i) applies*

Title and one or two sentences per requirement. Don’t repeat requirements implemented for prior assignments unless they are necessary to provide context—in which case, make it clear they are old.

### 4.2.1 Persist Imported Image Metadata into a Permanent Storage

The application persists the image metadata into a permanent storage during import and index them using the filename without extension. Any import that will have a file of similar name that exists already in the permanent storage will be overwritten with the new information from the imported file. The common fields like original src path, image type (JPG or specific camera manufacturer RAW type) and some EXIF field will have its own fields. In addition, the metadata will be stored in a serializable format so it can be easily hydrated by the application into an object as it bootstraps.

### 4.2.2 Retrieve Image Metadata of Images from Catalog from a Permanent Storage

The application will load the image metadata by name from the permanent storage and will use the serializable format to deserialize end create the corresponding object which makes it a lot more efficient to render the user interface.

### 4.2.3 Persist Image Preview of Imported Images into a Permanent Storage

The application persists the preview of the images into permanent storage during import and this is done and only done during import so that subsequent retrieval and rendering of the preview will be faster.

### 4.2.4 Retrieve Preview of Imported Images from a Permanent Storage

The application retrieves the preview of imported images that’s stored in the permanent storage by name and be able to hold in memory throughout the duration of the life of the application’s runtime.

### 4.2.5 Render Preview of Imported Images into a Permanent Storage

Requirement 3.2.2 is “Render the RAW Image to the User Interface” which involves reading the entire original image file to be read every time which would be inefficient and time-consuming. This approach will allow us to not only read the bytes from a database but also render a much smaller image which will make the application snappier.

### 4.2.6 Import Images from the GUI

The application should allow the user to (interactively) import directory of images by prompting a directory selection dialog box and if confirmed, the app will start the import operation in the same manner as if the import operation is launched from the command line. Selecting cancel should abort the operation.

## 4.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE

*Evaluation criterion (ii) applies*

Provide an example of input / output showing the new features of your application.

### Metadata and Preview Persistence

The serializable format to be used is JSON and we are using Jackson to serialize/deserialize the ImageMetadata object. For permanent storage we are using SQLite database in a table called . Further information on the use of Jackson and SQLite (along with the database schema) can be found in the README file.

As you import a directory of image files the metadata and preview are generated and as they are written to the database the metadata is serialized into JSON and displayed on the log file.

The images are indexed by name (the base file name without extension) and each image will have a row with the common attributes, the metadata JSON and the preview (as a blob).

### Input / Output

In the source code under /logs the import.log file is an example logs as generated during the import of the files from a directory.

And here is a snippet of it (it’s very long):

|  |
| --- |
| 10:18:47.183 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - Connecting to jdbc:sqlite:db/jlitebox.sqlite  10:18:47.407 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - Connecting to jdbc:sqlite:db/jlitebox.sqlite  10:18:47.409 [JavaFX Application Thread] INFO edu.bu.cs622.jlitebox.AppFx - Importing from directory sample/images  10:18:47.409 [JavaFX Application Thread] INFO e.b.c.j.v.ImageImporterConsoleView - Importing from path: sample/images  10:18:47.409 [JavaFX Application Thread] INFO e.b.c.j.image.BasicImageCatalog - Adding images from dir sample/images  10:18:47.428 [JavaFX Application Thread] INFO e.b.c.j.image.BasicImageCatalog - L1007408 is a JPG  10:18:47.429 [JavaFX Application Thread] INFO e.b.c.j.image.BasicImageCatalog - R0000102 is a JPG  ...  10:18:47.429 [JavaFX Application Thread] INFO e.b.c.j.image.BasicImageCatalog - 030327\_172925\_0325 is a CRW  10:18:47.439 [JavaFX Application Thread] INFO edu.bu.cs622.jlitebox.image.RawImage - Generating Image Preview for DSCF3887  10:18:49.345 [JavaFX Application Thread] INFO e.b.c.j.i.p.LibRawImagePreviewGenerator - Preview generated width: 640.0, height: 425.0  ...  10:18:55.618 [JavaFX Application Thread] INFO e.b.c.j.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-04/sample/images/L1007408.JPG  10:18:55.632 [JavaFX Application Thread] INFO e.b.c.j.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-04/sample/images/R0000102.JPG  10:18:55.640 [JavaFX Application Thread] INFO e.b.c.j.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-04/sample/images/DSCF3887.RAF  10:18:55.662 [JavaFX Application Thread] INFO e.b.c.j.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-04/sample/images/B0000747.JPG  10:18:55.672 [JavaFX Application Thread] INFO e.b.c.j.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-04/sample/images/L1007905.JPG  10:18:55.675 [JavaFX Application Thread] INFO e.b.c.j.storage.FileImageStorage - Copying image /Users/dexter/Projects/CS622/bu-ms-s5-04/sample/images/DSC\_8465.NEF  ...  10:18:55.816 [JavaFX Application Thread] INFO e.b.c.j.i.m.LibRawMetadataExtractor - Metadata successfully extracted from L1007408; it has 33 metadata tags  10:18:55.828 [JavaFX Application Thread] INFO e.b.c.j.i.m.LibRawMetadataExtractor - Metadata successfully extracted from R0000102; it has 33 metadata tags  10:18:55.977 [JavaFX Application Thread] INFO e.b.c.j.i.m.LibRawMetadataExtractor - Metadata successfully extracted from DSCF3887; it has 33 metadata tags  10:18:55.979 [JavaFX Application Thread] INFO e.b.c.j.i.m.LibRawMetadataExtractor - Metadata successfully extracted from B0000747; it has 33 metadata tags  10:18:55.980 [JavaFX Application Thread] INFO e.b.c.j.i.m.LibRawMetadataExtractor - Metadata successfully extracted from L1007905; it has 33 metadata tags  ...  10:18:57.646 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - Saving metadata for L1007408  10:18:57.776 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - JSON for L1007408: {"rawData":{"CameraModel":"M (Typ 240)","MaxAp @MaxFocal":"f/16.0","ExposureProgram":"-1","MedteringMode":"-1","Focal length":"35.0 mm","Shutter":"29.999998","MaxFocal":"35.0 mm","Timestamp (EpocheSec)":"1522888967","Aperture":"f/1.4","InternalLensSerial$slice":"","GPS Altitude":"0.0","MaxAp @MinFocal":"f/1.4","ExposureMode":"-1","ImageWidth":"0","GPS Position":"0;0","CurFocal":"0.0","DriveMode":"-1","ImageStabiMode":"-1","CameraMaker":"Leica","shutter":"0.033333335","FocalLengthIn35mmFormat":"0 mm","AFPoint":"-1","LensSerial$slice":"","LensMake":"Leica Camera AG","GPS Time":"","CurAperture":"0.0","ImageHeight":"0","MaxAperture @CurFocal":"f/1.3989797","ImageColors":"0","ISO speed":"1000.0","Lens":"Summilux-M 1:1.4/35 ASPH.","FocusMode":"-1","MinFocal":"35.0 mm"},"aperture":1.4,"shutterSpeed":29.999998,"captureDate":1522888967000,"iso":1000}  10:18:57.782 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - L1007408 metadata and preview successfully saved to database  10:18:57.787 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - Saving metadata for R0000102  10:18:57.800 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - JSON for R0000102: {"rawData":{"CameraModel":"GR IIIx","MaxAp @MaxFocal":"f/0.0","ExposureProgram":"-1","MedteringMode":"-1","Focal length":"26.0 mm","Shutter":"499.99997","MaxFocal":"0.0 mm","Timestamp (EpocheSec)":"1634398600","Aperture":"f/5.6","InternalLensSerial$slice":"","GPS Altitude":"0.0","MaxAp @MinFocal":"f/0.0","ExposureMode":"-1","ImageWidth":"0","GPS Position":"0;0","CurFocal":"0.0","DriveMode":"-1","ImageStabiMode":"-1","CameraMaker":"Ricoh","shutter":"0.002","FocalLengthIn35mmFormat":"40 mm","AFPoint":"-1","LensSerial$slice":"","LensMake":"","GPS Time":"","CurAperture":"0.0","ImageHeight":"0","MaxAperture @CurFocal":"f/0.0","ImageColors":"0","ISO speed":"200.0","Lens":"","FocusMode":"-1","MinFocal":"0.0 mm"},"aperture":5.6,"shutterSpeed":499.99997,"captureDate":1634398600000,"iso":200}  10:18:57.801 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - R0000102 metadata and preview successfully saved to database  ...  10:18:57.954 [JavaFX Application Thread] INFO edu.bu.cs622.jlitebox.AppFx - Import from directory complete  Process finished with exit code 0 |

The screen below shows the contents of the SQLite table after ingestion. Note that the raw\_metadata holds the serialized JSON:

|  |
| --- |
| **select name**, **camera\_model**, **camera\_brand**, **raw\_metadata from** image\_metadata; |

Graphical user interface, text, application

Description automatically generated

What is probably worth noting here is that in IntelliJ when you query the table, the image blob can be viewed directly as an image as shown in the screenshot below:

|  |
| --- |
| Graphical user interface, text, application, table  Description automatically generated |

Like the previous iteration of the application, we are still able to render the imported image, but the difference here is that the image preview is loaded from the database and no longer extracted from the original image which makes it a lot faster:

|  |
| --- |
| Graphical user interface, website  Description automatically generated |

In addition, starting in this iteration you don’t have to use the command line option to import images from a directory; you can do this from the GUI by selecting File 🡪 Import Images… from the menu bar. It should be noted that the import here is done in the background so the GUI remains responsive during import and automatically refreshes once the import is done. The same logging can be seen as if the import is done via the command line.

## 4.4 YOUR DIRECTORY

Show a screenshot of your directory. Include your “.dat” files (where objects are written). This should include JUnit tests—class-by-class, and method-by-method, except for trivial and inappropriate ones.

The database is under the /db folder

|  |  |
| --- | --- |
| A picture containing graphical user interface  Description automatically generated | Graphical user interface, application  Description automatically generated with medium confidence |
| Graphical user interface, application  Description automatically generated | Graphical user interface, application  Description automatically generated |

## 4.5 DESIGN

*Evaluation criterion (i) applies*

Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red your class model where you applied object read, object write, streams and lambdas. Excellent assignments will typically include the use of Java FX (speak to your facilitator first if you wish to use alternative API’s) and event-driven programming.

### 4.5.1 Use Case: Saving Metadata and Preview to Permanent Storage

This is the main use case that deals with persisting the metadata and preview after it’s been extracted from the image to be imported.

#### Updated Class Diagram

The class diagram largely remained unchanged because the class DatabaseImageStorage (where Object serialization and persistence is done) has been defined previously but was mostly mocked until this iteration. It has also been updated to clearly identify the generic interfaces.

Diagram

Description automatically generated

Larger version can be seen [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-642%2C-34%2C2049%2C1170%2C7AaROlDR.kas&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

#### Sequence Diagram

This is the sequence diagram from User clicking on the “Import…” dialog down to saving the metadata and preview in the database

Diagram

Description automatically generated

Larger version can be seen [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-642%2C-34%2C2049%2C1170%2C7AaROlDR.kas&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

## 4.6 CODE SNIPPETS

*Evaluation criterion (iii) applies*

### 4.6.1 Code showing object read and write

The DatabaseImageStorage is where the object read/write happens. As noted earlier, we are not using binary serialization but instead using JSON serialization via Jackson. We are using ObjectMapper to serialize/deserialize the ImageMetadata into a java String (using DatabaseImageStorage::toJson method) and back (i.e., we are not hand-building the JSON String) and this same JSON is saved into database as a text field:

|  |
| --- |
| public final class DatabaseImageMetadataStorage  implements ImageMetadataStorage, Closeable {  private static Logger logger = LoggerFactory  .getLogger(DatabaseImageMetadataStorage.class);  // the Jackson serializer  private final ObjectMapper mapper;  // the image preview generator class  private final ImagePreviewGenerator previewGenerator;  // the SQLite db connection  Connection conn;  @Inject  public DatabaseImageMetadataStorage(  ImageCatalogConfiguration config,  ImagePreviewGenerator previewGenerator)  throws SQLException {  this.mapper = new ObjectMapper();  this.previewGenerator = previewGenerator;  logger.info("Connecting to {}", config.getDatabaseUrl());  this.conn = DriverManager.getConnection(config.getDatabaseUrl());  }  // this serializes the metadata object into String  private String toJson(ImageMetadata metadata) {  return Try.of(() ->  mapper.writeValueAsString(metadata)).getOrNull();  }  // rest of the class definition ...  } |

This is where the code is persisted in database (note the use of try/catch with resources construct PreparedStatement). Note that normally for databases we use a JPA layer (like Hibernate) and/or use Stored Procedures. However, with SQLite the existing JPA implementation can be (i.e., current Hibernate dialect for SQLite is not even version 1.0) and also, with the way it works (i.e., it’s a file and not a client/server) it cannot support stored procedures.

|  |
| --- |
| // this is the "upsert" statement; if the image name already exists in the  // database, we just update the metadata hence the ON CONFLICT(name) clause  static final String UPSERT\_PSTATEMENT = "insert into image\_metadata(name, src\_path, image\_type, camera\_brand, " +  "camera\_model, camera\_autofocus," +  "lens\_brand, lens\_model, lens\_focal\_length, shutter\_speed, " +  "capture\_date, iso, image\_preview, raw\_metadata) " +  "values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?) on conflict(name) do update " +  "set src\_path = excluded.src\_path, " +  "image\_type = excluded.image\_type, " +  "camera\_brand = excluded.camera\_brand, " +  "camera\_model = excluded.camera\_model, " +  "camera\_autofocus = excluded.camera\_autofocus, " +  "lens\_brand = excluded.lens\_brand, " +  "lens\_model = excluded.lens\_model, " +  "lens\_focal\_length = excluded.lens\_focal\_length, " +  "shutter\_speed = excluded.shutter\_speed, " +  "capture\_date = excluded.capture\_date, " +  "iso = excluded.iso, " +  "image\_preview = excluded.image\_preview, " +  "raw\_metadata = excluded.raw\_metadata";  /\*\*  \* save to database  \*  \* @param image the image  \* @param previewBytes the byte array for the preview  \* @param jsonMetadata metadata in json format  \* @return return false if failed  \*/  private boolean saveToDatabase(Image image, byte[] previewBytes, String jsonMetadata) {  try (var pstmt = conn.prepareStatement(UPSERT\_PSTATEMENT) {  var meta = image.getMetadata();  pstmt.setString(1, image.getName());  pstmt.setString(2, image.getOriginalSrcPath());  pstmt.setString(3, image.getType());  pstmt.setString(4, meta.getCamera().map(Camera::getBrand).orElse(null));  pstmt.setString(5, meta.getCamera().map(Camera::getModel).orElse(null));  pstmt.setInt(6, meta.getCamera().map(c -> c.isAutofocus() ? 1 : 0).orElse(0));  pstmt.setString(7, meta.getLens().map(Lens::getBrand).orElse(null));  pstmt.setString(8, meta.getLens().map(Lens::getModel).orElse(null));  pstmt.setFloat(9, meta.getLens().map(Lens::getFocalLength).orElse((float) 0));  pstmt.setFloat(10, meta.getShutterSpeed());  pstmt.setInt(11, (int) meta.getCaptureDate().getTime());  pstmt.setInt(12, meta.getIso());  pstmt.setBytes(13, previewBytes);  pstmt.setString(14, jsonMetadata);  logger.info("{} metadata and preview successfully saved to database", image.getName());  pstmt.execute();  } catch (SQLException e) {  logger.error("Error in saving metadata for {}: {}", image.getName(), e.getMessage(), e);  }  return true;  } |

And this is the code that retrieves JSON and other metadata from the database. We are using an intermediate (internal) class DatabaseRecord to get the raw data prior to hydration:

|  |
| --- |
| /\*\*  \* internal class for loading the data from database  \*/  static class DatabaseRecord {  byte[] previewBytes;  String jsonMetadata;  Image image;  DatabaseRecord(String srcPath, String imgType, byte[] previewBytes, String jsonMetadata) {  this.image = imgType.equalsIgnoreCase(ImageUtils.JPG) ? new RawImage(srcPath) : new JpegImage(srcPath);  this.jsonMetadata = jsonMetadata;  this.previewBytes = previewBytes;  }  }  static final String QUERY\_ONE\_IMAGE\_PSTATEMENT =  "select src\_path, image\_type, raw\_metadata, image\_preview from image\_metadata where name = ?";  /\*\*  \* Load from the database. Returns the Image with (partial) metadata along with  \* the preview bytes and json metadata  \*  \* @param name the name to look up  \* @return image metadata record from db  \*/  private Optional<DatabaseRecord> loadFromDatabase(String name) {  try (var stmt = conn.prepareStatement(QUERY\_ONE\_IMAGE\_PSTATEMENT) {  stmt.setString(1, name);  var rs = stmt.executeQuery();  if (rs.next()) {  // pull the columns  var src = rs.getString(1);  var itype = rs.getString(2);  var jsonMetadata = rs.getString(3);  var previewBytes = rs.getBinaryStream(4).readAllBytes();  var rec = new DatabaseRecord(src, itype, previewBytes, jsonMetadata);  return Optional.of(rec);  } else {  return Optional.empty();  }  } catch (SQLException e) {  logger.error("Unable to retrieve {} from db due to SQL error: {}",  name, e.getMessage(), e);  return Optional.empty();  } catch (IOException e) {  logger.error("Unable to retrieve {} from db due to I/O error: {}",  name, e.getMessage(), e);  return Optional.empty();  }  } |

And finally, how the metadata is hydrated using the ObjectMapper::readValue (which takes the class type ImageMetadata.class so it knows to construct ImageMetadata from the JSON:

|  |
| --- |
| @Override  public Optional<Image> loadImage(String name) {  var rec = loadFromDatabase(name).map(r -> {  try {  // deserialize the metadata from JSON to ImageMetadata class using Jackson  var metadata = mapper.readValue(r.jsonMetadata, ImageMetadata.class);  // create the preview image from the bytes from blob in record  var preview = previewGenerator.createPreviewFromByteArray(r.previewBytes);  r.image.setMetadata(metadata);  r.image.setPreview(preview);  return r.image;  } catch (JsonProcessingException e) {  logger.error("Error loading JSON metadata for {}", name, e);  return null;  } catch (ImageImportException e) {  logger.error("Error loading preview for {}", name, e);  return null;  }  });  logger.info("Successfully loaded from db: {}", name);  return rec;  } |

### 4.6.2 Code Showing *stream*() and Lambdas

There are several places where the streams and lambdas are used. One notable use of this to process the files in the directory for imported files in BasicCatalogImage::addImagesFromDirectory, here namely to filter them based on the supported file types/extensions loaded from configuration:

|  |
| --- |
| @Override  public int addImagesFromDirectory(String srcDir, ImageImportOptions options) throws JLiteBoxException {  logger.info("Adding images from dir {}", srcDir);  var dir = new File(srcDir);  var newImages = FileUtils.listFiles(dir,  config.supportedFileExtensions().toArray(new String[0]), false)z.stream()  .map(f -> ImageFactory.fromFile(f.getAbsolutePath()))  .collect(Collectors.toList());  return addImages(newImages, options);  } |

Simpler stream/lambda constructs are also for generating the previews for the images:

|  |
| --- |
| newImages.stream()  .filter(i -> i instanceof RawImage)  .forEach(i -> Try.run(() -> ((RawImage) i).generateImagePreview(getPreviewGenerator()))); |

For lambdas, specifically, this is used in ContentImageFilter::test. The use of lambda essentially allows them to be able to be composed and in this case, we are applying [currying](https://en.wikipedia.org/wiki/Currying) show how the functions drive the behavior of filtering.

|  |
| --- |
| @Override  public boolean test(Image image) {  // the idea here is that image metadata is tested against the filter where the  // filter field is not Optional.empty()  // if the filter field exists and matches it will return true  var meta = image.getMetadata();  // matching of fields; we are using lambdas here to be more explicit  // what fields we are matching and to avoid repetition of code  // in the next section; also for readability  Function<CharSequence, Boolean> nameMatch = v -> image.getName().contains(v);  Function<CharSequence, Boolean> fileExtMatch = v -> image.getFilename().contains(v);  Function<CharSequence, Boolean> cameraMatch = v -> meta.getCamera()  .map(c -> c.getBrand().contains(v))  .orElse(false);  Function<CharSequence, Boolean> lensMatch = v -> meta.getLens()  .map(l -> l.getBrand().contains(v))  .orElse(false);  Function<Float, Boolean> focalLengthMatch = v -> meta.getLens()  .map(l -> l.getFocalLength() == v)  .orElse(false);  Function<Float, Boolean> shutterSpeedMatch = v -> meta.getShutterSpeed() == v;  Function<Float, Boolean> apertureMatch = v -> meta.getAperture() == v;  Function<Integer, Boolean> isoMatch = v -> meta.getIso() == v;  if (isAndOperation()) {  // all match  return name.map(nameMatch).orElse(true)  && fileExt.map(fileExtMatch).orElse(true)  && cameraName.map(cameraMatch).orElse(true)  && lensName.map(lensMatch).orElse(true)  && focalLength.map(focalLengthMatch).orElse(true)  && shutterSpeed.map(shutterSpeedMatch).orElse(true)  && aperture.map(apertureMatch).orElse(true)  && iso.map(isoMatch).orElse(true);  } else {  // any match  // since Java 9 .or() is composable and this is better than in previous  // versions of Java otherwise we will have to use the construct above  // similar to AND  return name.map(nameMatch)  .or(() -> fileExt.map(fileExtMatch))  .or(() -> cameraName.map(cameraMatch))  .or(() -> lensName.map(lensMatch))  .or(() -> focalLength.map(focalLengthMatch))  .or(() -> shutterSpeed.map(shutterSpeedMatch))  .or(() -> aperture.map(apertureMatch))  .or(() -> iso.map(isoMatch))  .orElse(false);  }  } |

## 4.7 YOUR CODE

[Here is a link](https://drive.google.com/file/d/1yYauYKhBBKGifMV_xIme1i-ZRYj8kvvN/view?usp=sharing) for the video walkthrough of this iteration of the project. Download the video (download icon at the upper right corner) to avoid the potato quality that Google Drive offers when it streams the video using the browser.

## 4.8 YOUR CODE

*Evaluation criterion (iii) applies*

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to your Blackboard response. Please contact your facilitator in advance if you require an exception.

Code is in a separate zip file.

## 4.9 Evaluation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criterion** | **D** | **C** | **B** | **A** | **Letter Grade** | **%** |
| **(i) 4.1 Summary Description and 4.2 Requirements and 4.5 Updated Class Model** | Hard to understand. | Understandable | … and concise and all requirements are declarative statements of functionality | … and extremely clear and complete functionality can be built from the requirement statements alone. | A+ | 97.0 |
| **(ii) 4.3 I/O evidence that the above functionality was achieved** | Minimal requirements met. | Some committed requirements are clearly met and a ReadMe file is included. | All committed requirements are met and they are non-trivial | … and in clear correspondence with all requirement details. | A+ | 97.0 |
| **(iii) 4.6 Techniques implemented and 4.7 Your code** | Design or code hard to understand | Implementation works and design is understandable, and code mostly understandable from comments | … and concise (no redundant parts) and appropriately applied and code understandable from comments | … and all methods precisely specified and very clear. | A- | 90.0 |
|  |  |  |  | Assignment Grade: |  | 94.7 |
|  |  |  |  |  |  |  |
| The resulting grade is the average of these, using A+=97, A=95, A-=90, B+=87, B=85, B-=80 etc. The maximum is 100. | | | | | |  |
| To obtain an A grade for the course, your weighted average should be >93. A-:>=90. B+:>=87. B:>83. B-:>=80 etc. | | | | | |  |

22/9/2021 11:55 AM

# Assignment 5

Implement the next release of your term project, employing *concurrency* if concurrency can fit; otherwise create a different project with concurrency. The same instructions as before apply to this completed Word document, the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file. As usual, copying someone else’s application without clear attribution is plagiarism, and will be subjected to the College’s academic conduct process.

## 5.1 SUMMARY DESCRIPTION

*Evaluation criterion (i) applies*

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from Assignment 2, if any (all in red if this is a separate application).

JLiteBox is an application to import digital camera images (JPG and various RAW formats; see [this explanation](https://helpx.adobe.com/lightroom-cc/how-to/raw-vs-jpeg.html) for the difference between the formats.) to store them in a common repository and be able to view them with simple and convenient navigation scheme akin to a virtual image lightbox. This is basically a scaled down version of [Adobe Lightroom](https://www.adobe.com/products/photoshop-lightroom.html) or [Camera Bits PhotoMechanic](https://home.camerabits.com/) but with just the browsing and tagging capabilities (i.e., no image editing capabilities)

It will be leveraging the [JavaFX](https://openjfx.io/index.html) libraries to create the GUI and be able to render images on the screen and navigate through them individually and also show them in a grid format and use existing open-source libraries to extract image metadata (e.g., camera/lens information). There will be a means to filter images and only show a subset of the images from the catalog in the grid. There is also an option to show or hide the metadata. The pertinent metadata will be stored in a local database for speed and efficiency. In addition, since generating a preview is an expensive process (which involves reading the original file and then generating the preview), the previews will be stored in the database along with the metadata, as reading the smaller binary stream will be much more efficient.

There will also be a toolbar that allows a few other additional options like changing the size of the previews. The toolbar can be toggled on/off in case the user needs the screen real estate.

Since the application is essentially a read only view of the image catalog, it will provide a means in its GUI to either show the file in the filesystem (where the originals are stored) similar to “Show in Finder” in MacOS and/or a means to launch the application associated with the file (usually this will be Adobe Photoshop)

There are also two options to import files without having to launch the GUI: one is using by specifying the directory and the other is specifying a file with a list of URLs where the application can download them and add to the catalog. It emits log messages as the image are downloaded and since network connections are more fragile than local directory, the application offers a way to recover if one file (to skip it) has an issue with downloading.

## 5.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

*Evaluation criterion (i) applies*

Title and one or two sentences per requirement. Don’t repeat requirements implemented for prior assignments unless they are necessary to provide context—in which case, make it clear they are old.

### 5.2.1 Display ALL the Images from the Catalog in a Grid

The application should allow to display all images in the catalog in a grid-like fashion with its corresponding metadata.

### 5.2.3 Optionally Hide Metadata in Grid

The application should provide an option to display or hide metadata when displaying in a grid via a Menu toggle option.

### 5.2.4 Optionally Show a Convenient Toolbar

The application should provide an option to display or hide metadata when displaying in a grid via a toolbar where user can easily access.

### 5.2.5 Optionally Show RAW Metadata on Image Mouse Hover

The application should provide an option to display RAW Metadata on an Image on mouse hover and can be disabled via a Menu toggle option.

### 5.2.5 Optionally Resize the Image Preview

The application should provide an option to resize the displayed preview by using a slider control.

### 5.2.6 Import Images from the Background without Blocking UI

The application should allow the user to (interactively) import directory of images by prompting a directory selection dialog box and if confirmed, the app will start the import operation in the same manner as if the import operation is launched from the command line. Selecting cancel should abort the operation. This reflects 4.2.6 except there’s an additional enhancement whereby the import is done in the background to allow the GUI to be still responsive and update the GUI with the status of the import

## 5.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE

*Evaluation criterion (ii) applies*

Provide an example of input / output showing the new features of your application.

### Input/Output

The GUI displaying the images of the catalog. (Animated GIF)

|  |
| --- |
|  |

See the GUI still being responsive while import is being done (shown with logging) and also updating the status bar (Animated GIF).

|  |
| --- |
|  |

You can toggle showing/hiding metadata from the View > Show/Hide Metadata menu:

Metadata Visible

|  |
| --- |
| Graphical user interface, website  Description automatically generated |

Metadata Hidden

|  |
| --- |
| Graphical user interface, website  Description automatically generated |

When mouse cursor is hovered over an image, a tooltip will appear showing the RAW metadata. It can be disabled via a menu option (Animated gif):

|  |
| --- |
| A picture containing text, screenshot  Description automatically generated |

Toolbar can be accessed using the menu which shows up on left side of the UI (Animated gif):

|  |
| --- |
|  |

Preview can be resized by using a slider control (Animated gif):

|  |
| --- |
|  |

## 5.4 YOUR DIRECTORY

Show a screenshot of your directory. Include your “.dat” files (where objects are written). This should include JUnit tests—class-by-class, and method-by-method, except for trivial and inappropriate ones.

The database is under the /db folder

|  |  |
| --- | --- |
| A picture containing graphical user interface  Description automatically generated | Graphical user interface, application  Description automatically generated with medium confidence |
| Graphical user interface, application  Description automatically generated | Graphical user interface, application  Description automatically generated |

## 5.5 DESIGN

*Evaluation criterion (i) applies*

Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red your class model where you applied object read, object write, streams and lambdas. Excellent assignments will typically include the use of Java FX (speak to your facilitator first if you wish to use alternative API’s) and event-driven programming.

The class diagram remains largely unchanged except for the callout for the MainController::handleImportDirectory method (Upper left corner) which is the menu item handler for “Import Images from Directory…

Diagram

Description automatically generated

Larger version can be seen [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-1693%2C-629%2C1772%2C1012%2C0_0&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

Similarly, the Sequence for Importing Files is mostly the same, except there is now a highlight where asynchronous call and parallelism is applied (in red)

Diagram

Description automatically generated

Larger version can be seen [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-879%2C-106%2C2049%2C1170%2C7AaROlDR.kas&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

## 5.6 CODE SNIPPETS

*Evaluation criterion (iii) applies*

A demonstration video explaining the use of concurrency can be viewed [here](https://youtu.be/DlQRf8hmJbQ).

### 5.6.1 Code showing where concurrency is *defined*

Two of the main uses of concurrency in the application is when the user Imports images from the directory: The import operation is launched asynchronously in MainController::handleFileImportAction in the background using CompletableFuture::runAsync so the GUI can remain responsive throughout the import operation.

|  |
| --- |
| @FXML  protected void handleFileImportAction(ActionEvent event) {  logger.info("Import...");  DirectoryChooser importDirDialog = new DirectoryChooser();  importDirDialog.setTitle("Import Images");  var selectedDirectory = importDirDialog  .showDialog(((MenuItem) event.getTarget()).getParentPopup().getOwnerWindow());  if (selectedDirectory != null) {  // runAsync to not block the GUI; JavaFx is single-threaded and only allows  // updated in its own thread (it doesn't allow update of GUI from another thread  // this is why there is reference to that use of Platform::runLater which  // allows asynchronous import \_but still\_ be able to update the GUI during the  // progress of import  statusBar.setText(  String.format("Importing images from directory [%s]...", selectedDirectory.getPath()));  CompletableFuture.runAsync(() -> Try.of(() -> catalog.addImagesFromDirectory(  selectedDirectory.getAbsolutePath(),  ImageFactory.withDefaultImportOptions(),  (currentImage, currentIndex, all) -> {  logger.info("Image {} of {} imported.", currentIndex, all);  // this allows to update the status bar in the UI on the  // status of import \_asynchronously\_  updateStatusBarText(  String.format("Image %d of %d imported.", currentIndex, all));  return null;  }))).thenRun(() -> Platform.runLater(() -> {  // once everything is finished, refresh the main view  logger.info("Refreshing view...");  initializeImageCollectionView();  statusBar.setText(statusBarDefaultText);  }));  }  } |

In the same import sequence flow in BasicImageCatalog::addImages, the images are imported *in parallel*; we are using streams and using combination of CompletableFuture::supplyAsync, CompletableFuture::theApplyAsync, and CompletableFuture::thenRun we are able to generate preview, extract metadata, and update the GUI status bar as 3 separate threads per file imported.

We are also using AtomicInteger for counting the processed images as they are being processed. Also note that we are using a custom thread pool here based on Executors::newFixedThreadPool because this is I/O heavy and [ForkJoinPools is usually not as efficient](https://stackoverflow.com/questions/8206318/is-javas-fork-and-join-thread-pool-is-good-for-executing-io-bound-task).

|  |
| --- |
| protected int addImages(List<Image> newImages,  ImageImportOptions options,  Function3<Image, Integer, Integer, Void> importCallback) throws JLiteBoxException {  this.images.addAll(newImages);  // list the types  newImages.forEach(i -> {  logger.info("{} is a {}", i.getName(), i.getType());  });  // Atomic integer is thread-safe and simple and allows us to use an incrementer  // without having to deal manually with mutexes, critical sections nor even  // deal explicitly with synchronized keyword. Also note that each thread here  // on separate files so there is no need to "lock" resources being mutated.  AtomicInteger counter = new AtomicInteger(0);  try {  imageStorage.save(newImages);  // We are using threads here to execute the processing of each file in parallel  // this benefits parallelism because each generate preview and read metadata  // operation operates on a different file and each process is expensive and  // heavy in I/O and having them execute in parallel allows us to be able  // to efficiently use multiple threads of execution and the import process  // finishes quicker.  CompletableFuture.allOf(newImages.stream().map(i -> CompletableFuture  // Note that each thread of execution is further broken  // down in to 3 threads of execution (one for creating preview, one for reading  // metadata, and one to update GUI status bar to allow yielding thread of  // execution automagically without having to specify Thread::yield  .supplyAsync(() -> Try.of(() -> generatePreviewForImage(i))  .onFailure(t ->  logger.warn("Failed to generate preview for {}",  i.getName())), threadPool)  .thenApplyAsync(ti -> ti.mapTry(metadataExtractor::parse)  .onFailure(t ->  logger.warn("Failed to extract metadata for {}",  i.getName()))  .andThen(mo -> mo.ifPresent(i::setMetadata)), threadPool)  .thenRun(() -> {  // this is executed in another thread for updating the UI  // but not block the next image to be processed.  if (importCallback != null)  importCallback.apply(i, counter.incrementAndGet(), newImages.size());  })).toArray(CompletableFuture[]::new)).join(); // wait for threads to finish  // this saves all metadata in the database at this point. SQLite operations are  // serialized into single thread even if we explicitly multi-thread it so we're  // not getting any performance advantage to using threads here.  imageMetadataStorage.saveMetadata(newImages);  } catch (JLiteBoxException e) {  if (!options.isIgnoreErrors())  throw e;  }  logger.info("Catalog has {} images", newImages.size());  return newImages.size(); } |

### 5.6.2 Code showing where concurrency is *used*

The code is used within the GUI. It is using FXML (main.fxml) to define the Main View, the Main View has a Menu and one of the items is “Import Images…” and when that is selected the handler assigned #handleFileImportAction calls MainController::handleFileImportAction. Note that the handleImportAction has @FXML annotation so the FXML file below can associate the “Import Directory…” menu item to the handleImportAction:

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>*  *<?*import javafx.scene.control.\**?>*  *<?*import javafx.scene.layout.\**?>*  *<?*import org.controlsfx.control.\**?>*  *<*BorderPane style="*-*fx-background-color: lightgray*"* fx:id="mainViewPane" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" minHeight="-Infinity" minWidth="-Infinity" xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1" fx:controller="edu.bu.cs622.jlitebox.controller.gui.MainController"*>*  *<*top*>*  *<*MenuBar*>*  *<*Menu text="File"*>*  *<*MenuItem onAction="#handleAboutDialog" text="About..." */>*  *<*MenuItem onAction="#handleFileImportAction" text="Import Images..." */>*  *<*SeparatorMenuItem */>*  *<*MenuItem onAction="#handleFileExitAction" text="Exit" */>*  *</*Menu*>*  *<*Menu text="View"*>*  *<*MenuItem onAction="#handleToggleShowMetadata" text="Hide Metadata" */>*  *</*Menu*>*  *</*MenuBar*>*  *</*top*>*  *<*center*>*  *<*VBox style="*-*fx-background-color: lightgray*"* fx:id="mainViewContainer" alignment="TOP\_CENTER" prefHeight="300.0" prefWidth="800.0"*>*  *<*ScrollPane style="*-*fx-background-color: lightgray*" >*  *<*TilePane style="*-*fx-background-color: lightgray*"* fx:id="gridContainer" hgap="10" prefColumns="8" vgap="10" */>*  *</*ScrollPane*>*  *</*VBox*>*  *</*center*>*  *<*bottom*>*  *<*StatusBar fx:id="statusBar" text="Ready." */>*  *</*bottom*>*  *</*BorderPane*>* |

## 5.7 YOUR CODE

*Evaluation criterion (iii) applies*

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to your Blackboard response. Please contact your facilitator in advance if you require an exception.

Code is in a separate zip file.

## 5.8 Evaluation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criterion** | **D** | **C** | **B** | **A** | **Letter Grade** | **%** |
| **(i) 5.1 Summary Description and 5.2 Requirements and 5.5 Updated Class Model** | Hard to understand. | Understandable | … and concise and all requirements are declarative statements of functionality | … and extremely clear and complete functionality can be built from the requirement statements alone. | A+ | 97.0 |
| **(ii) 5.3 I/O evidence that the above functionality was achieved** | Minimal requirements met. | Some committed requirements are clearly met and a ReadMe file is included. | All committed requirements are met and they are non-trivial | … and in clear correspondence with all requirement details and the application of concurrency is entirely appropriate. | A+ | 97.0 |
| **(iii) 5.6 Techniques implemented and 5.7 Your code** | Design or code hard to understand | Implementation works and design is understandable, and code mostly understandable from comments | … and concise (no redundant parts) and appropriately applied and code understandable from comments | … and all methods precisely specified and very clear. | A+ | 97.0 |
|  |  |  |  | Assignment Grade: |  | 100.0 |
|  |  |  |  |  |  |  |
| The resulting grade is the average of these, using A+=97, A=95, A-=90, B+=87, B=85, B-=80 etc. The maximum is 100. | | | | | |  |
| To obtain an A grade for the course, your weighted average should be >93. A-:>=90. B+:>=87. B:>83. B-:>=80 etc. | | | | | |  |

# Assignment 6 2/16/2021 12:04 PM

Implement the next release of your term project, employing a database manipulated from your application.

If you want to use a database other than SQLite as described, please check with your facilitator.

The same instructions as previously apply to this completed Word document, the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

# 6.1 SUMMARY DESCRIPTION

*Evaluation criterion (i) applies*

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from Assignment 4.

JLiteBox is an application to import digital camera images (JPG and various RAW formats; see [this explanation](https://helpx.adobe.com/lightroom-cc/how-to/raw-vs-jpeg.html) for the difference between the formats.) to store them in a common repository and be able to view them with simple and convenient navigation scheme akin to a virtual image lightbox. This is basically a scaled down version of [Adobe Lightroom](https://www.adobe.com/products/photoshop-lightroom.html) or [Camera Bits PhotoMechanic](https://home.camerabits.com/) but with just the browsing and tagging capabilities (i.e., no image editing capabilities)

It will be leveraging the [JavaFX](https://openjfx.io/index.html) libraries to create the GUI and be able to render images on the screen and navigate through them individually and also show them in a grid format and use existing open-source libraries to extract image metadata (e.g., camera/lens information). There will be a means to filter images and only show a subset of the images from the catalog in the grid based on the image metadata and the UI will provide a means to indicate if the filtering is logical OR or AND. There is also an option to show or hide the metadata. The pertinent metadata will be stored in a local database for speed and efficiency. In addition, since generating a preview is an expensive process (which involves reading the original file and then generating the preview), the previews will be stored in the database along with the metadata, as reading the smaller binary stream will be much more efficient.

There will also be a toolbar that allows a few other additional options like changing the size of the previews. The toolbar can be toggled on/off in case the user needs the screen real estate.

Since the application is essentially a read only view of the image catalog, it will provide a means in its GUI to either show the file in the filesystem (where the originals are stored) like “Show in Finder” in MacOS and/or a means to launch the application associated with the file (usually this will be Adobe Photoshop)

There are also two options to import files without having to launch the GUI: one is using by specifying the directory and the other is specifying a file with a list of URLs where the application can download them and add to the catalog. It emits log messages as the image are downloaded and since network connections are more fragile than local directory, the application offers a way to recover if one file (to skip it) has an issue with downloading.

There is also now an option to export the catalog as a JSON file. This JSON file contains a JSON array of individual images in the catalog along with their metadata.

# 6.2 KEY REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

*Evaluation criterion (i) applies*

Title and one or two sentences per requirement. Repeat requirements implemented for prior assignments if they are necessary to provide context. Make it clear which requirements are new vs. old.

### 6.2.1 Navigate Individual Image in the Catalog with a Larger Preview

The application allows you to view a larger preview of an image in the catalog and navigate through them individually.

### 6.2.2 Filter the Grid with Images that Meet Specified Filter

By specifying keywords you can filter the grid with a subset of it for easier management of the UI.

### 6.2.3 Open an Image with Default App

When viewing an individual image, there will be an option to open the original file with the operating system’s default application associated with the image type—e.g., for MacOS the JPEGs are launched with the Preview app.

### 6.2.4 Show Catalog Statistics

The application user interface shows some statistics about the catalog, e.g., the total number of images and the number of images meeting specific criteria.

### 6.2.4 Export Catalog as JSON

The application allows exporting the catalog of all the images metadata in a JSON array format and save it in a file selected by the user.

# 6.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE

*Evaluation criterion (ii) applies*

Provide an example of input / output showing the new features of your application.

A video with the details on how the new features is implemented can be found in this [video](https://youtu.be/I5EUsYb6L-o).

A video outlining the details of meeting the HW6 requirements can be found in this [video](https://youtu.be/QzvH2TO7YO0).

### Input/Output

As noted in the video, there are three tables (image, camera, lens) that the application writes data to and there exists a view (image\_with\_metadata). For the creation tables, it will be shown in the logs as the SQL scripts are executed.

|  |
| --- |
| 13:16:03.927 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - Connecting to jdbc:sqlite:db/jlitebox.sqlite  13:16:04.205 [JavaFX Application Thread] INFO e.b.c.j.s.DatabaseImageMetadataStorage - Initializing database...  -- tables for JLiteBox  -- dlegaspi@bu.edu  -- camera information  create table if not exists camera  (  id text  primary key,  brand text not null,  model text default 'Mansur',  is\_autofocus integer default 0  )  -- lens information  create table if not exists lens  (  id text  primary key,  brand text not null,  model text  )  -- the image  create table if not exists image  (  name text primary key,  image\_type text default 'JPG',  src\_path text not null,  camera\_id text,  lens\_id text,  lens\_focal\_length real,  shutter\_speed real,  capture\_date integer,  iso integer,  raw\_metadata text,  image\_preview blob,  foreign key (camera\_id) references camera (id),  foreign key (lens\_id) references lens (id)  )  create view if not exists image\_with\_metadata as  select i.\*,  c.brand as camera\_brand,  c.model as camera\_model,  c.is\_autofocus as camera\_autofocus,  l.brand as lens\_brand,  l.model as lens\_model  from image i  left join camera c on i.camera\_id = c.id  left join lens l on i.lens\_id = l.id  13:16:04.214 [JavaFX Application Thread] INFO e.b.c.j.c.gui.SplashController - Launching main GUI controller... |

# 6.4 YOUR DIRECTORY

Show a screenshot of your directory. Include all relevant files. This should include JUnit tests.

Nothing notable has changed in this iteration of the homework.

|  |  |
| --- | --- |
| A picture containing graphical user interface  Description automatically generated | Graphical user interface, application  Description automatically generated with medium confidence |
| Graphical user interface, application  Description automatically generated | Graphical user interface, application  Description automatically generated |

# 6.5 DESIGN

*Evaluation criterion (i) applies*

### 6.5.1 Class Model, Use Case, and Sequence Diagram

## Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red where you applied the features listed below.

The only notable change in the diagram is the addition of 2 methods in ImageMetadataStorage to support the export of catalog as JSON getAllAsList and statistics with getStatistics

Diagram

Description automatically generated

The larger version can be found [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

The sequence diagram showed below is for “Export Catalog as JSON” where the application can export the metadata of all images in the catalog as a JSON array in a file specified by the user.

Diagram

Description automatically generated

The larger version can be found [here](https://lucid.app/lucidchart/3c247b44-8ed2-4389-9308-9dc9cbf0bf56/edit?viewport_loc=-973%2C-108%2C2269%2C1296%2C7AaROlDR.kas&invitationId=inv_772c48aa-eb77-4fff-adeb-a585065eda38).

# 6.6 CODE SNIPPETS

*Evaluation criterion (iii) applies*

### 6.6.1 Code of any kind showing table creation

.The application uses a combination of Java code and SQL code for table creation. It using ScriptRunner to run a script and the script is in a separate file

The SQL script. Note the use of if not exist clause so it doesn’t re-create the tables (and deleting the existing data) if they already exist.

|  |
| --- |
| -- camera information  create table if not exists camera  (  id text  primary key,  brand text not null,  model text,  is\_autofocus integer default 0  );  -- lens information  create table if not exists lens  (  id text  primary key,  brand text not null,  model text  );  -- the image  create table if not exists image  (  name text primary key,  image\_type text default 'JPG',  src\_path text not null,  camera\_id text,  lens\_id text,  lens\_focal\_length real,  shutter\_speed real,  capture\_date integer,  iso integer,  raw\_metadata text,  image\_preview blob,  foreign key (camera\_id) references camera (id),  foreign key (lens\_id) references lens (id)  );  create view if not exists image\_with\_metadata as  select i.\*,  c.brand as camera\_brand,  c.model as camera\_model,  c.is\_autofocus as camera\_autofocus,  l.brand as lens\_brand,  l.model as lens\_model  from image i  left join camera c on i.camera\_id = c.id  left join lens l on i.lens\_id = l.id; |

The code

|  |
| --- |
| @Override  public void initialize() {  logger.info("Initializing database...");  ScriptRunner sr = new ScriptRunner(conn);  sr.runScript(new InputStreamReader(getClass().getResourceAsStream("/db/jlitebox-tables.sql")));  try {  conn.close();  this.conn = createConnection();  } catch (SQLException e) {  logger.error("Error in initialization {}", e.getMessage(), e);  }  } |

### 6.6.2 Java code involving the insertion of data

.We are using PreparedStatements for data insertion

|  |
| --- |
| // this is the "upsert" statement; if the image name already exists in the  // database, we just update the metadata hence the ON CONFLICT(name) clause  static final String UPSERT\_PSTATEMENT\_IMAGE = "insert into image(name, src\_path, image\_type, " +  "camera\_id, lens\_id, lens\_focal\_length, shutter\_speed, " +  "capture\_date, iso, image\_preview, raw\_metadata) " +  "values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?) on conflict(name) do update " +  "set src\_path = excluded.src\_path, " +  "image\_type = excluded.image\_type, " +  "camera\_id = excluded.camera\_id, " +  "lens\_id = excluded.lens\_id, " +  "lens\_focal\_length = excluded.lens\_focal\_length, " +  "shutter\_speed = excluded.shutter\_speed, " +  "capture\_date = excluded.capture\_date, " +  "iso = excluded.iso, " +  "image\_preview = excluded.image\_preview, " +  "raw\_metadata = excluded.raw\_metadata";  // camera  static final String UPSERT\_PSTATEMENT\_CAMERA = "insert into camera(id, brand, model, is\_autofocus) " +  "values (?, ?, ?, ?) on conflict (id) do update " +  "set brand = excluded.brand, model = excluded.model, is\_autofocus = excluded.is\_autofocus";  // lens - note that focal length is NOT on the lens because it could be a zoom  // lens  static final String UPSERT\_PSTATEMENT\_LENS = "insert into lens(id, brand, model) " +  "values (?, ?, ?) on conflict (id) do update " +  "set brand = excluded.brand, model = excluded.model"; |

Below is how it they are used:

|  |
| --- |
| private boolean saveToDatabase(Image image, byte[] previewBytes, String jsonMetadata) {  var meta = image.getMetadata();  // camera  var camera = meta.getCamera();  camera.ifPresent(c -> {  try (var pstmt = conn.prepareStatement(UPSERT\_PSTATEMENT\_CAMERA)) {  pstmt.setString(1, c.getId());  pstmt.setString(2, c.getBrand());  pstmt.setString(3, c.getModel());  pstmt.setInt(4, c.isAutofocus() ? 1 : 0);  pstmt.execute();  logger.info("{} camera info successfully saved to database", image.getName());  } catch (SQLException e) {  logger.error("Error in saving camera metadata for {}: {}",  image.getName(), e.getMessage(), e);  }  });  // lens  var lens = meta.getLens();  lens.ifPresent(l -> {  try (var pstmt = conn.prepareStatement(UPSERT\_PSTATEMENT\_LENS)) {  pstmt.setString(1, l.getId());  pstmt.setString(2, l.getBrand());  pstmt.setString(3, l.getModel());  pstmt.execute();  logger.info("{} lens info successfully saved to database", image.getName());  } catch (SQLException e) {  logger.error("Error in saving lens metadata for {}: {}",  image.getName(), e.getMessage(), e);  }  });  // save the image  try (var pstmt = conn.prepareStatement(UPSERT\_PSTATEMENT\_IMAGE)) {  pstmt.setString(1, image.getName());  pstmt.setString(2, image.getOriginalSrcPath());  pstmt.setString(3, image.getType());  pstmt.setString(4, meta.getCamera().map(Camera::getId).orElse(null));  pstmt.setString(5, meta.getLens().map(Lens::getId).orElse(null));  pstmt.setFloat(6, meta.getLens().map(Lens::getFocalLength).orElse((float) 0));  pstmt.setFloat(7, meta.getShutterSpeed());  pstmt.setLong(8, meta.getCaptureDate() != null ? meta.getCaptureDate().toInstant().toEpochMilli() / 1000  : (new Date()).toInstant().toEpochMilli() / 1000);  pstmt.setInt(9, meta.getIso());  pstmt.setBytes(10, previewBytes);  pstmt.setString(11, jsonMetadata);  pstmt.execute();  logger.info("{} metadata and preview successfully saved to database", image.getName());  } catch (SQLException e) {  logger.error("Error in saving image metadata for {}: {}", image.getName(), e.getMessage(), e);  }  return true;  } |

### 6.6.3 Java code involving the selection of records, and ordering with SQL

|  |
| --- |
| static final String QUERY\_IMAGES\_PSTATEMENT = "select src\_path, image\_type, raw\_metadata " +  "from image\_with\_metadata order by capture\_date desc";  @Override  public List<Image> getAllAsList() throws ImageOperationException {  var list = new ArrayList<Image>();  try (var stmt = conn.prepareStatement(QUERY\_IMAGES\_PSTATEMENT)) {  var rs = stmt.executeQuery();  while (rs.next()) {  var src = rs.getString(1);  var itype = rs.getString(2);  var jsonMetadata = rs.getString(3);  var rec = new DatabaseRecord(src, itype, null, jsonMetadata);  loadImage(rec.image.getName()).ifPresent(list::add);  }  } catch (SQLException e) {  throw new ImageOperationException("Unable to retrieve metadata list due to DB error: " + e.getMessage(), e);  }  return list;  } |

### 6.6.4 Java code involving selection involving at least two tables

As noted we are using a view to do a select, so this is implicitly met. Here is the definition of that view once more:

|  |
| --- |
| create view if not exists image\_with\_metadata as  select i.\*,  c.brand as camera\_brand,  c.model as camera\_model,  c.is\_autofocus as camera\_autofocus,  l.brand as lens\_brand,  l.model as lens\_model  from image i  left join camera c on i.camera\_id = c.id  left join lens l on i.lens\_id = l.id; |

### 6.6.5 Java code involving selection and aggregation of data

This is displayed in Catalog Statistics in the UI

|  |
| --- |
| private String getStatsFromQuery(@NonNull String query) {  return Try.of(() -> {  var s = conn.createStatement();  var rs = s.executeQuery(query);  if (rs.next()) {  return rs.getString(1);  } else {  return "0";  }  }).onFailure(t -> logger.warn("Unable to get stats due to error: {}", t.getMessage(), t))  .getOrElse("-");  }  @Override  public Map<String, String> getStatistics() {  // we are using LinkedHashMap to maintain order the way they are inserted  var map = new LinkedHashMap<String, String>();  map.put("Images", getStatsFromQuery("select count(\*) from image"));  map.put("JPEG Images", getStatsFromQuery("select count(\*) from image where image\_type = 'JPG'"));  map.put("RAW Images", getStatsFromQuery("select count(\*) from image where image\_type <> 'JPG'"));  map.put("Cameras", getStatsFromQuery("select count(\*) from camera"));  map.put("Lenses", getStatsFromQuery("select count(\*) from lens"));  map.put("Taken with AF cameras",  getStatsFromQuery("select count(\*) from image\_with\_metadata where camera\_autofocus = 1"));  return map;  } |

# 6.7 YOUR CODE

*Evaluation criterion (iii) applies*

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to the Blackboard submission. Please contact your facilitator in advance if you want to request an exception.

Code is in a separate zip file.

# 6.9 Evaluation

