|  |
| --- |
|  |
| Software Design Principles and Patterns(SWEN 383) |
| Image Album Project - Part 1 |
| Team 3 - Dominik Cvjetković, Mateo Mustapić, Antonio Šutalo |
|  |
|  |

|  |
| --- |
|  |

Contents

[Classes 3](#_Toc447490135)

[Libraries 3](#_Toc447490136)

[Responsibilities and Collaborators 4](#_Toc447490137)

[Diagrams 5](#_Toc447490138)

[Overview Diagram 5](#_Toc447490139)

[Subsystem 5](#_Toc447490140)

[Object Diagram 6](#_Toc447490141)

[Sequence Diagram 7](#_Toc447490142)

[Rationale 7](#_Toc447490143)

# Classes

In order to implement this project we have decided on having four separate classes where each class handles different responsibilities in order to create a high cohesion environment and promote the SoC design principle.

The classes we aim to use are:

* IMG Album Class
  + This is a controller class used to run the application. It creates and stores all objects created by using the other classes as well as displays the images for the user to interact with. Essentially this class behaves like a facade for the entire application.
* IMG Class
  + Class used to create image objects which stores all relevant data about each picture. It will be comprised of the standard get and set methods.
* IMG Loader Class
  + The loader class plays the role of finding images based on either file name, album or image tags. Class also creates image objects and loads appropriate data into them and returns these image objects to the controller class (IMG Album) for storage.
* IMG Editor Class
  + Editor class handles any image editing that is required by the user. In order to do this it takes an image object that needs to be edited from the controller class as well as the information related to the type of editing and afterwards returns the altered image object to the controller.

## Libraries

In order to achieve the full functionality that we have intended for our classes we will need to implement the Command Interface as well as use utilities such as Java 2D. At this point in time before we start exploring the full functionality of our code we cannot predict how many other non common libraries will be included.

## Responsibilities and Collaborators

|  |  |
| --- | --- |
| IMG Album | |
| Knows the IMG Loader  Knows all IMG objects  Knows the IMG Editor  Knows all commands used on a current image  Demands loading and creation of images  Demands changing of images  Displays images | IMG Loader  IMG  IMG Editor  Command Interface |

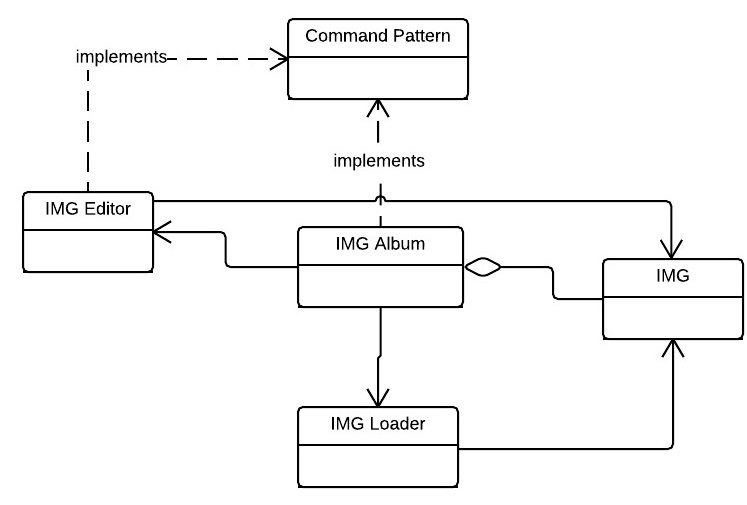
|  |  |
| --- | --- |
| IMG Loader | |
| Knows a possible file name/tag/album name  Discovers images based on possible file name/tag/album name  Finds out file name/tag/album name of a discovered image  Creates a new image object  Returns the new image object to the IMG Album | IMG  IMG Album |

|  |  |
| --- | --- |
| IMG | |
| Knows image file name  Knows image name  Knows date/time created/edited  Knows all image tags  Knows all image albums  Can change all information if needed  Can return all information if needed | IMG Album  IMG Loader  IMG Editor |

|  |  |
| --- | --- |
| IMG Editor | |
| Knows the image that is being edited  Knows the command being used for editing  Edits the image | IMG Album  Command Interface |

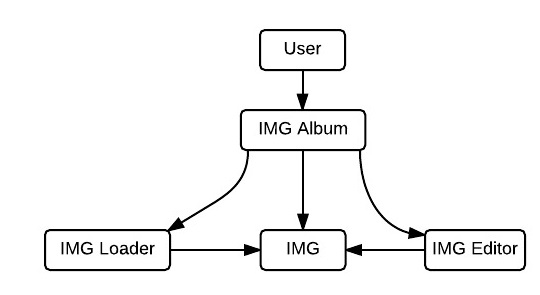
# Diagrams

## Overview Diagram



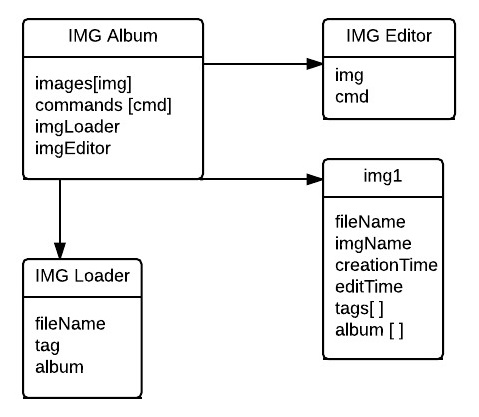
## Subsystem

The diagram depicts the intended use of the IMG Album class in the role of a facade for the application. The user will have access to it but without it the user would be unable to directly interact with any other classes. This however is a modified version of the facade pattern due to the fact that the IMG Album can interact with all parts of the subsystem while IMG Loader and IMG Editor can interact with the third part of the subsystem, the IMG object but they are still unaware of each other's existence.



## Object Diagram

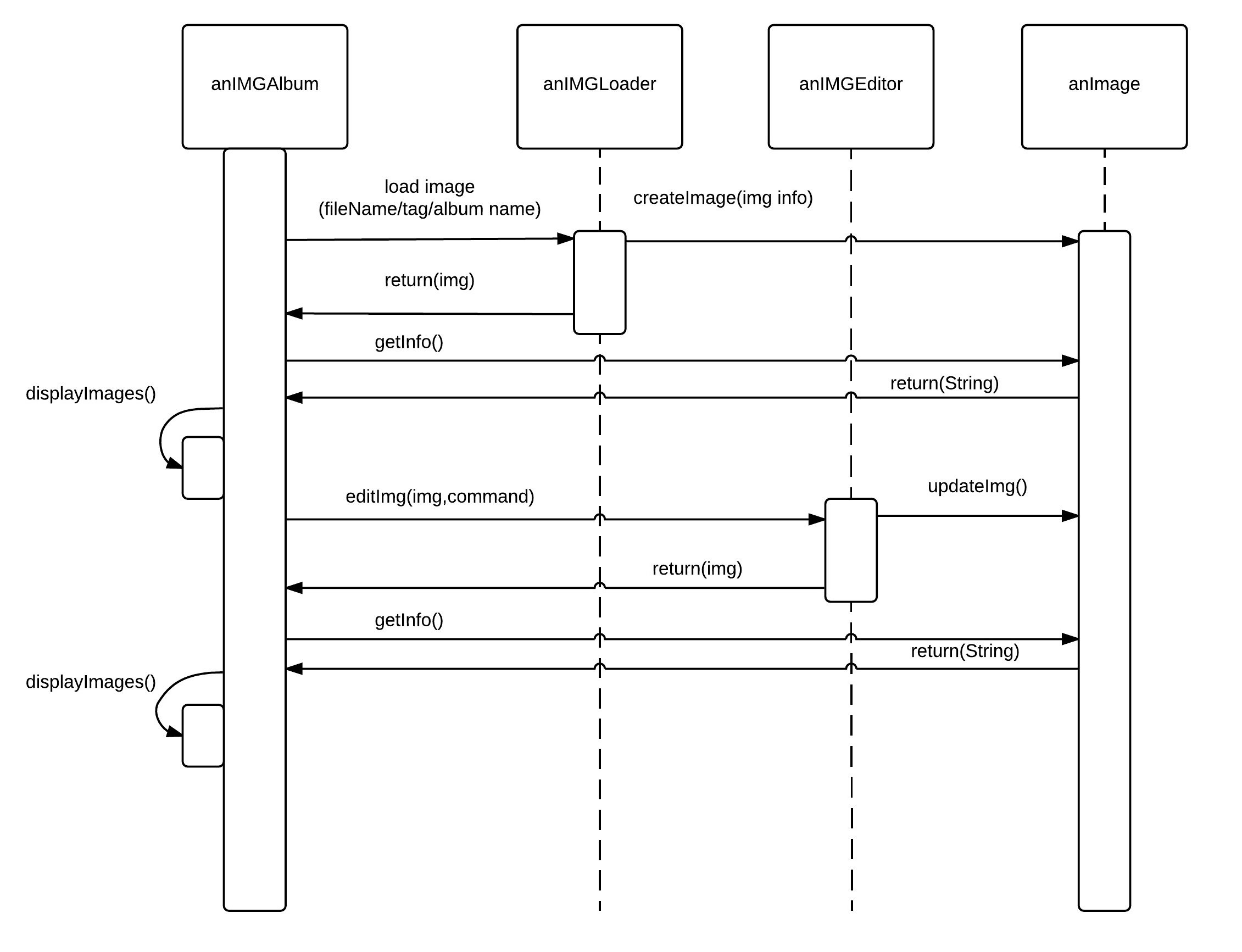
During run-time the application will consist of the IMG Album object which is created when the application starts. When needed the IMG Loader object will be created and it will be discarded after it has served its purpose. Same goes for the IMG Editor object. This approach was chosen to simplify the procedure of storing information about image editing in order to avoid conflicts in regards to how each image was edited. Furthermore during the run-time IMG objects will be created for every image that is used in an album. These image objects will persist as long as the album persists.



## Sequence Diagram

The sequence diagram is showing the handling of one of the most common procedures of the application - loading the files. This process would occur whenever the user loads images to an album.

The IMG Album class starts the IMG Loader class and uses one of its methods to load the image that user has wanted provided with either a file name, a tag or an album name. The IMG Loader class after finding the image(s) in question creates one or more image objects with the information it has discovered about them and returns all of the images to the IMG Album. The IMG Album class gets all relevant information for the image and then uses that information (title, caption, tags) to update the GUI and show the images. Once the user wants to edit an image the IMG Album class starts the IMG Editor and informs it about the image object that needs to be edited as well as the command which needs to be executed. The IMG Editor updates the image in question after the modifications are done and returns the new image back to the IMG Album which proceeds to check for new information about the image and updates the GUI and show the edited image.



## Rationale

We believe that an approach similar to the facade pattern would be the best choice for this application for the sake of the SoC principle. Even though the design in its current form is not a completely accurate depiction of the Facade pattern it still retains some of its strong suits which allows us to maintain a system in which most of the problems would be easy to track down. IMG Album class which is our representation of the facade class controls all parts of the subsystem directly and maintains a reference to all objects that exist within it while displaying information to the user. Methods invoked by this class as shown in the table of responsibilities consist of tasks such as requests for image loading or editing as well as get methods for information retrieval from images and the displaying of the same.

Furthermore we believe that the best way to keep track of all of the changes done to an image would be by storing all of the changes that have been done on an image in form of a command object. The command pattern would be used to hold commands such as resize, crop, flip, mirror and rotate. All of these commands would store any information needed to perform a modification but they would also hold information regarding the previous state of an image. This way we can maintain a reference for all of the changes and the exact manner in which they have been done.

With all of this in mind we agreed that this usage of the command pattern coupled with the facade pattern would be the best course of action for this application.