**Members**
Nikola Lenert
Tomislav Kulušić
Frano Nola
Pieter Rudoviq

# Image Album Project

**1st April 2017**

## Class Table, naming and describing each primary class in the design

### Class Responsibilities

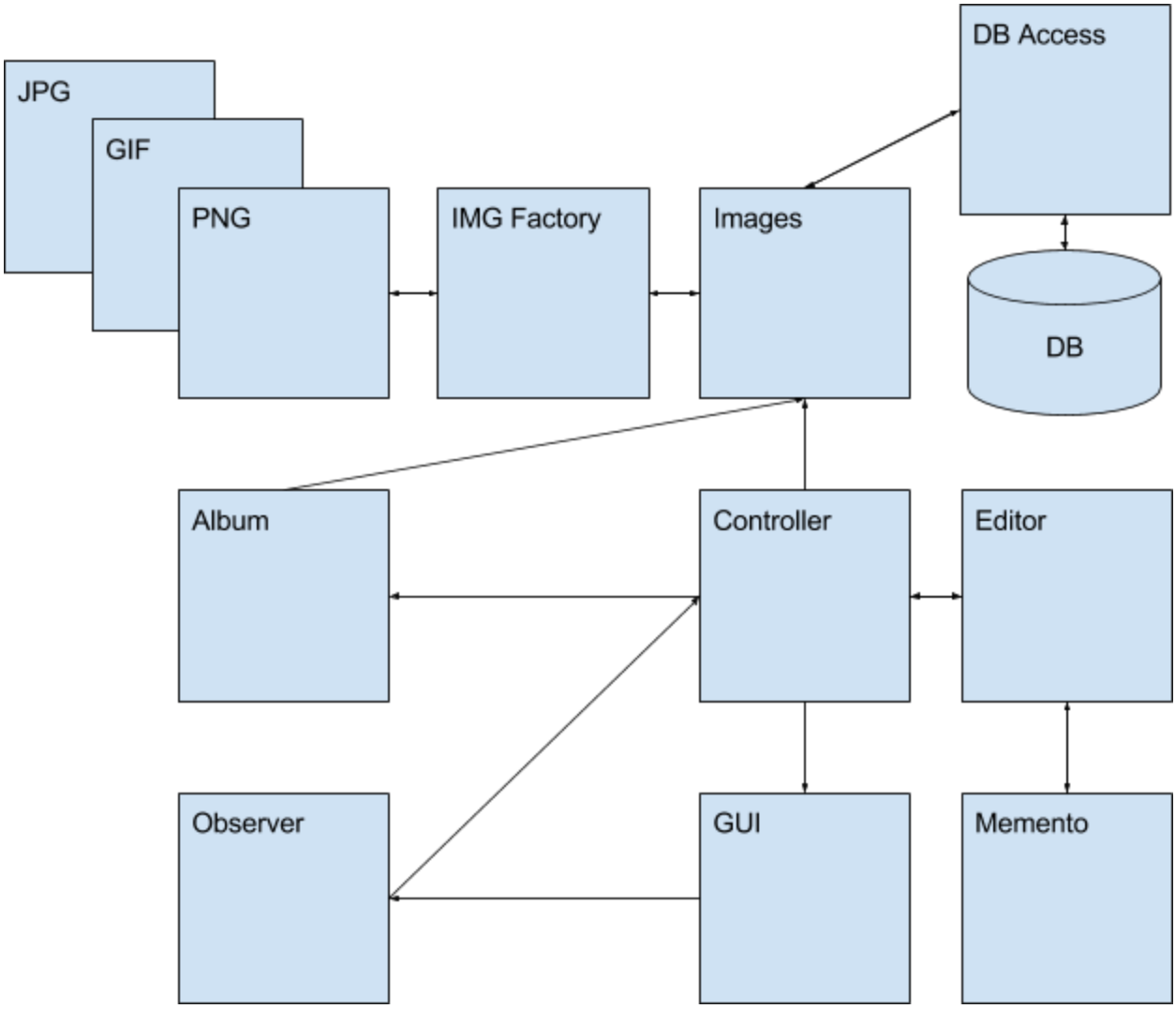
- GUI Class
    - Takes care of UI
    - Notifies Observer class of user interactions
    - Displays images, albums
- Images Class
    - Takes list of images out of database and stores image objects in array
- Album Class
    - Creates album object
    - Album object contains images
- Controller Class
    - Main controller class used to run application
    - Behaves like a facade
- Editor Class
    - Handles image editing
- DB Access
    - Accesses the database

## Class Collaborators

- GUI
  - Observer
    - Notifies observer of change
- Controller
  - GUI
    - Changes information displayed
  - Editor
    - Sends Image object to Editor to edit it
  - Album
    - Takes album information and sends it to gui on request
  - Images
    - Takes specific image on request
    - Adds image
- Album
  - Images
    - Takes image object from images and adds it to array
- Images
  - IMG Factory
    - Sends information about image and gets image object in return
  - DB Access
    - Requests all the images (their info, ex. tags) stored in database

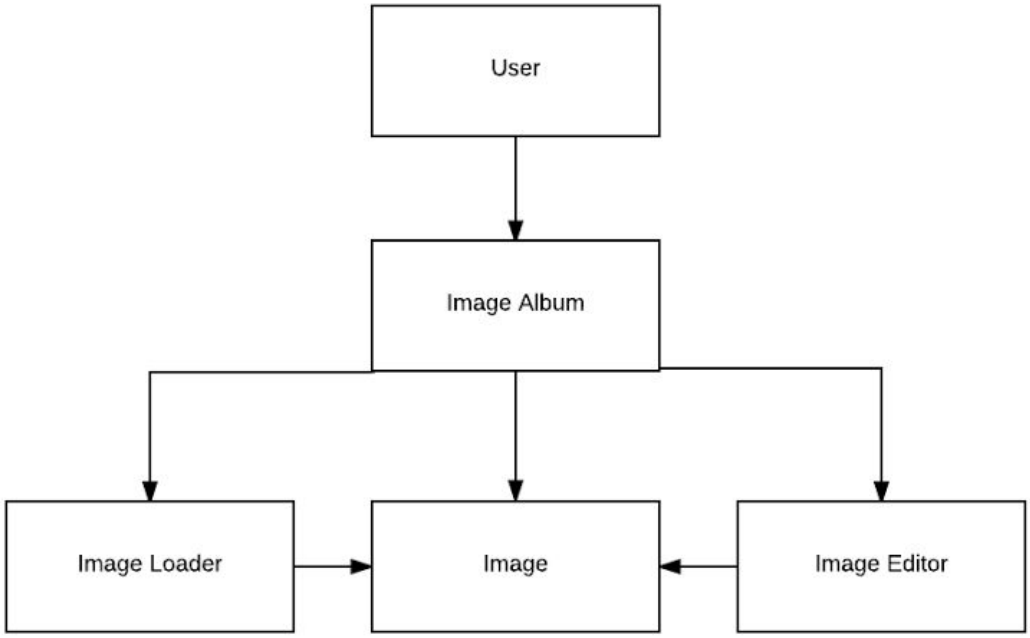
## UML Diagrams

### Overview Diagram



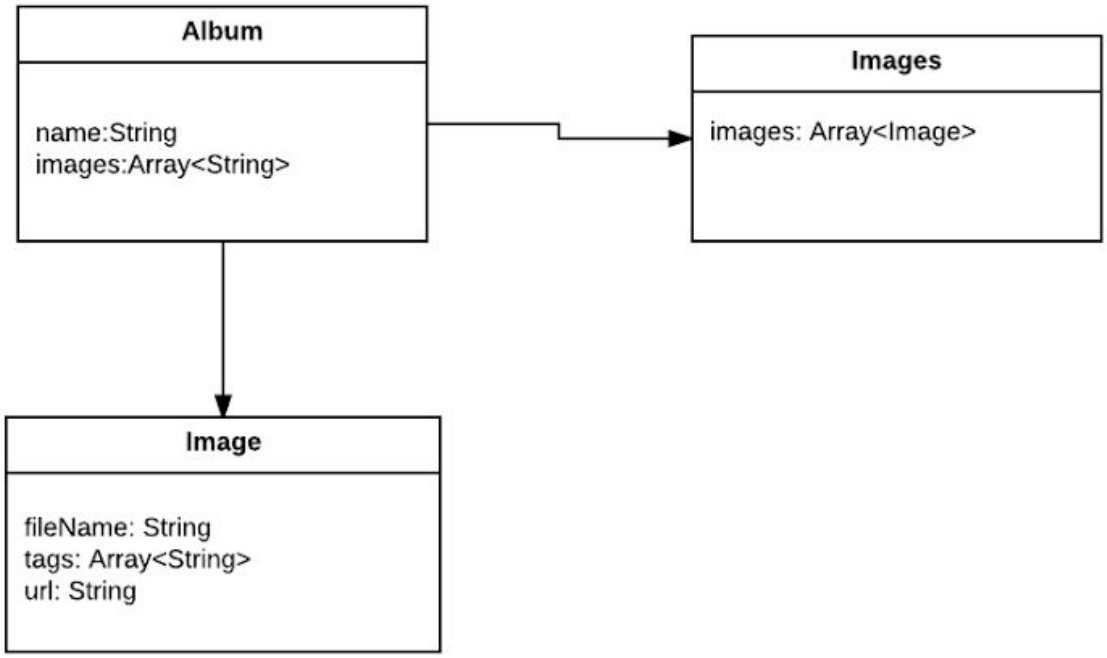
## Subsystem Diagrams

This diagram is showing the use of the Image Album Class in a role of the modified facade pattern. While user has access to Album Class he won't be able to access the other classes that communicates with Album. When i said modified facade pattern i mean that all parts of subsystem will be able to communicate with Image object, for example editor will be able to communicate with image and loader will communicate with Image as well.



## Object Diagrams

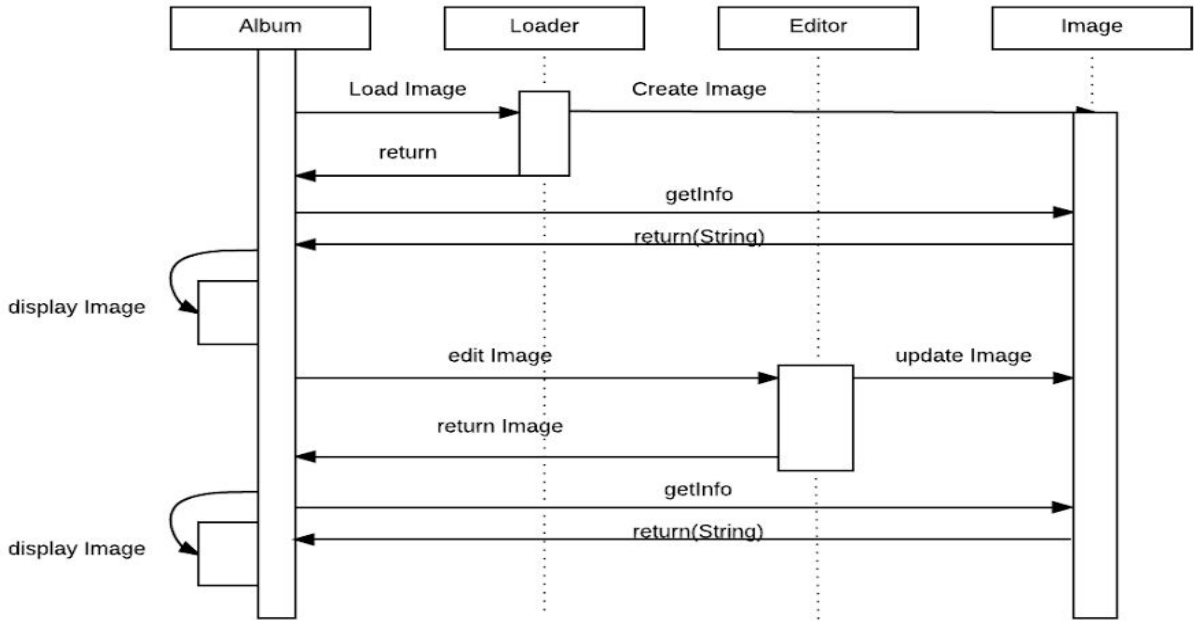
During the run the application will consist of the Album object which is created when the application starts. The Images object will load all the images and it will communicate with all objects needed for editing the picture. Object of the Image will always be created for every image used in Album.



## Sequence Diagrams

The sequence diagram is showing the one of the most common procedures of the application and that is loading the files. This process will occur whenever the user loads the images to the album.

The Album class will start the Loader class and it will use one of the methods to load the image depending on the filename, tag, or album name. When loader find the image depending on the user choice, it will return all the images to the Album. The Album class will get all the necessary informations required to update the view. When the user decides to update the image, the Editor class will start and it will inform user with all commands that he can use to edit the picture. The Editor will update the picture after the user is done with editing and it will return the new image to the album  and album will again update the GUI to show edited image.



## Rationale - decisions involving patterns

The first decision we made for this project is that we would use the Factory Method pattern. The reason for this is because in the image editor, we will be manipulating files of different extensions, such as .jpg, .png and .gif. We want different types of files to have different properties and information.

Another important pattern in our project is the Observer pattern. We will use it to connect the GUI buttons and functionalities to the methods that the buttons should execute. This way we are decoupling the GUI class from the actual functions of the program.

Composite pattern will be used because of the ability to load up albums in the program. We would like to enable the user to remove and add images to the album. Other functions using this pattern might be implemented as well.

Since we are required to implement the function of undoing and redoing actions when editing the image, we concluded that the best way to do this would be by implementing the Memento pattern. In order for Memento to properly track the object's state, we will combine it with the Command pattern.

The Command pattern will be responsible for holding the editing functions such as resizing, flipping, rotating and cropping the image, and it will also be able to hold the previous actions of the same function, which is vital for Memento to work.