horizontal line

**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**](#_9oj9l5uqxy5n) **2**

[Class Responsibilities](#_a85eciiq8lah) 2

[Class Collaborators](#_ig1q3buga74a) 3

[Libraries](#_dkp6ngi6y58v) 4

[**UML Diagrams**](#_c5rpsdy8g2ak) **4**

[Overview Diagram](#_mi96n59t87zs) 4

[Subsystem Diagrams](#_49lotq8csg7h) 5

[Object Diagrams](#_mk4qcdbhhd9l) 6

[Sequence Diagrams](#_5jfmo539g5xm) 7

[Rationale - decisions involving patterns](#_4jjl4ni7smg) 8

# 

# 

# 

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

## Class Responsibilities

* Loader
  + Loader class takes care of chosen image files. It will create an array of files from the directory that the user chose and in collaboration with the ImageCreator class it will form Image objects.
* ImageCreator
  + Class responsible for storing variables that an Image contains, together with their setters and getters.
* Album
  + Class holding Image objects, able to get specific images from the array of images. Decouples image selection from the data class.
* GUI
  + The presentational part of the application. JFrame instantiated here with all its elements that make up a GUI.
* Controller
  + Controller class handles user interaction with the GUI. Controller knows which button the user pressed and acts accordingly using the Action class.
* Action + Commands
  + The Action and its commands contain the code that are responsible for making the buttons do what they are supposed to, meaning cropping the image, rotating, resizing, etc...

## 

## 

## Class Collaborators

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

The classes that we are planing to use:

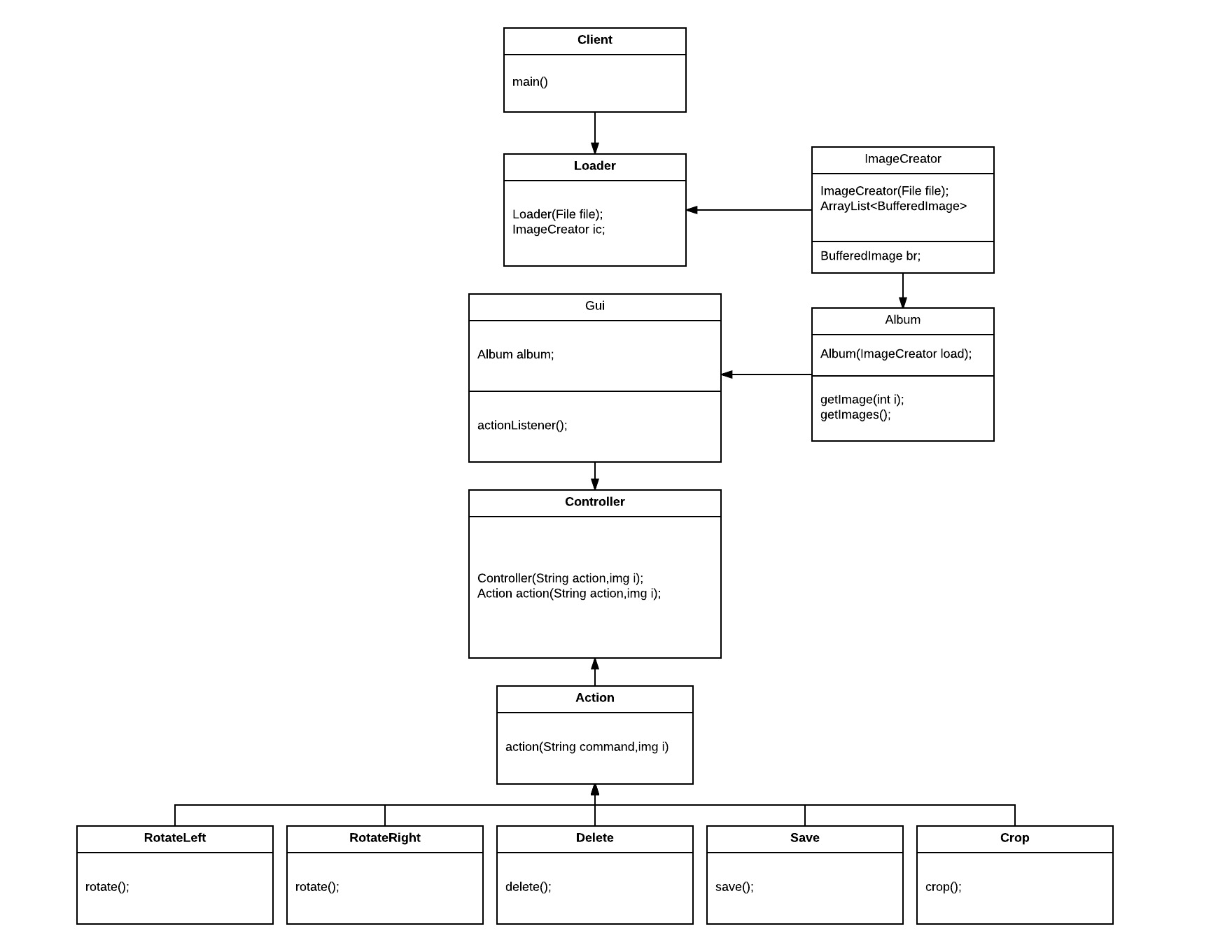
* Client
  + Client is where the main method of the program is located. It collaborates with the loader class which will take care of the loading of the images that the user will be editing.
* Loader
  + Loader class takes care of chosen image files, and collaborates with the ImageCreator class in order to create Image objects which are then stored into an array for easier access and manipulation.
* ImageCreator
  + The class where the attributes of the image file are stored. Collaborates with the Loader as described above and with the Album class in order to decouple image selection from the ImageCreator class.
* GUI
  + This is where all the presentational part of the code is handled. The class collaborates with the Album in order to provide the GUI with the actual images. It also collaborates with the Controller in order to enable the functionality of image editing, which is taken care of in the controller.
* Controller
  + Controller class will take care of the functionalities and user interaction with the GUI. Observer listens for changes on the GUI and notifies the controller which acts accordingly. Controller collaborates with the Action class which serves as an interface for all the possible actions/methods that can be carried out by the user on the image.
* Action + Commands
  + The classes where the actual code for what which action does is implemented. The Action class collaborates with controller in order to send the code back up to the GUI.

## Libraries

No libraries will be used in order to implement the functional part of the program. We might implement libraries for the looks of the application.

# UML Diagrams

## Overview Diagram

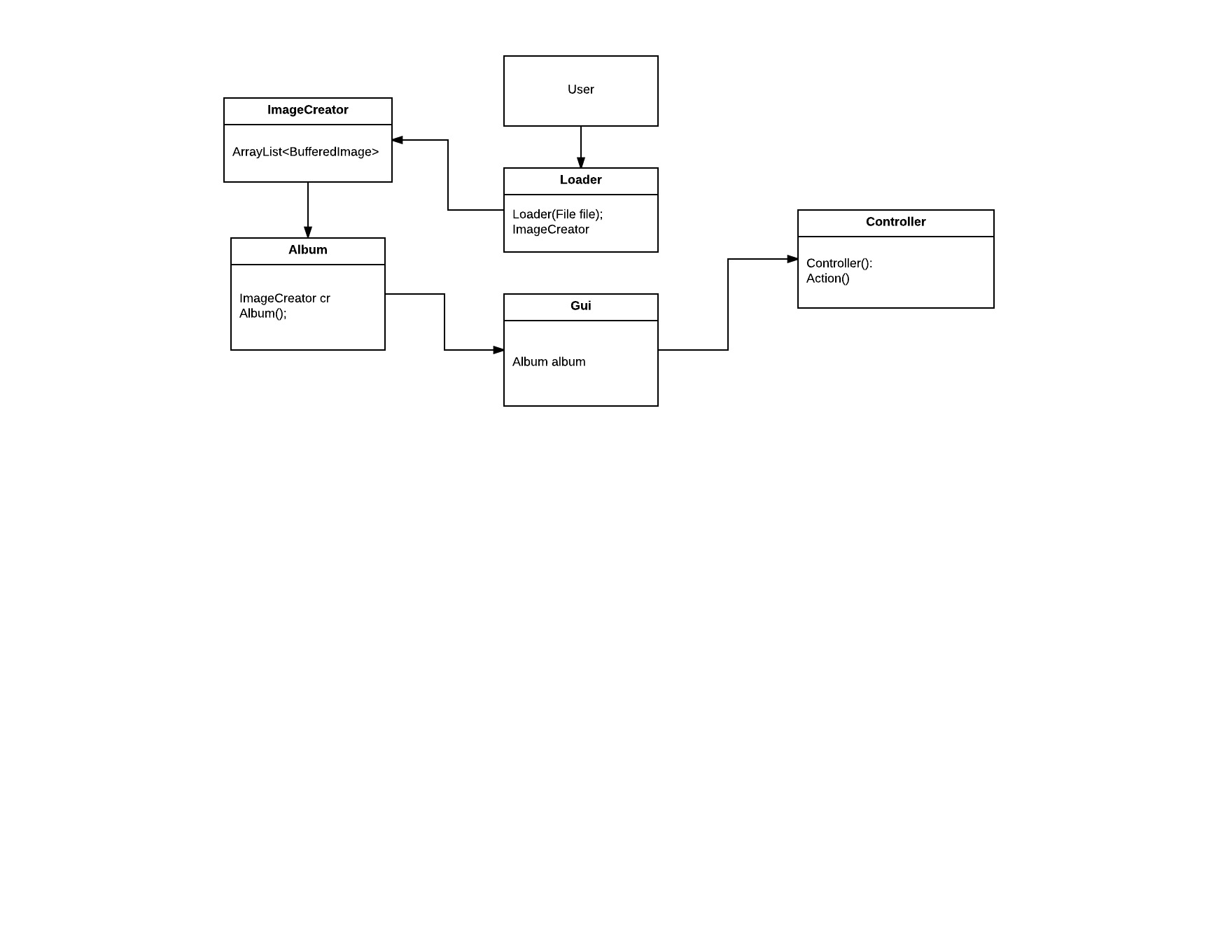


## 

## 

## 

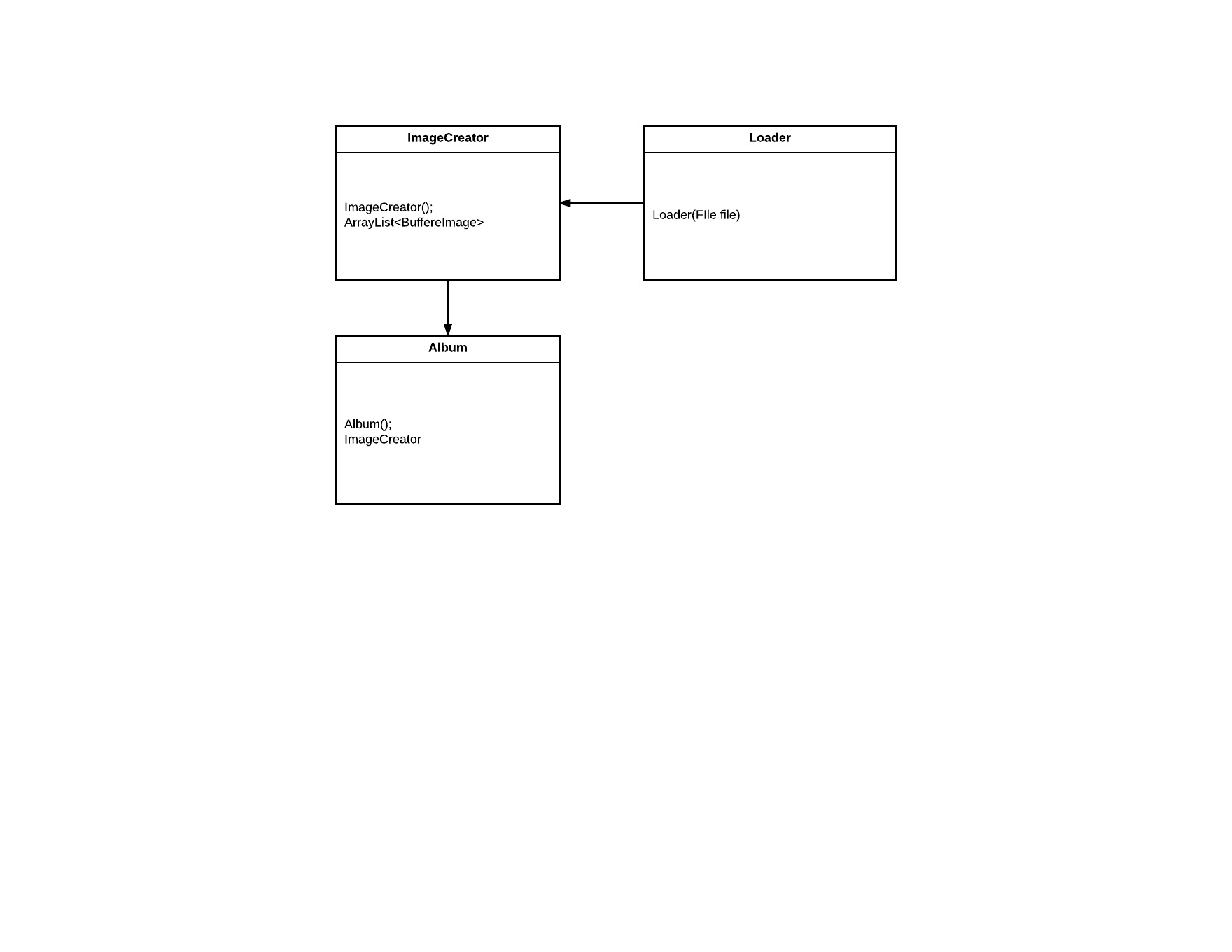
## Subsystem Diagrams

This diagram is showing the basic flow of the application. When user runs the application the images will be loaded and created using BufferedImage object in separate class and then they will be passed to the album class. Also here we have small example of MVC pattern where gui is playing the role of view and controller will be there to do all the actions. And also the controller is acting as some sort of facade pattern. Behind him there is class for every action.

## 

## 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.

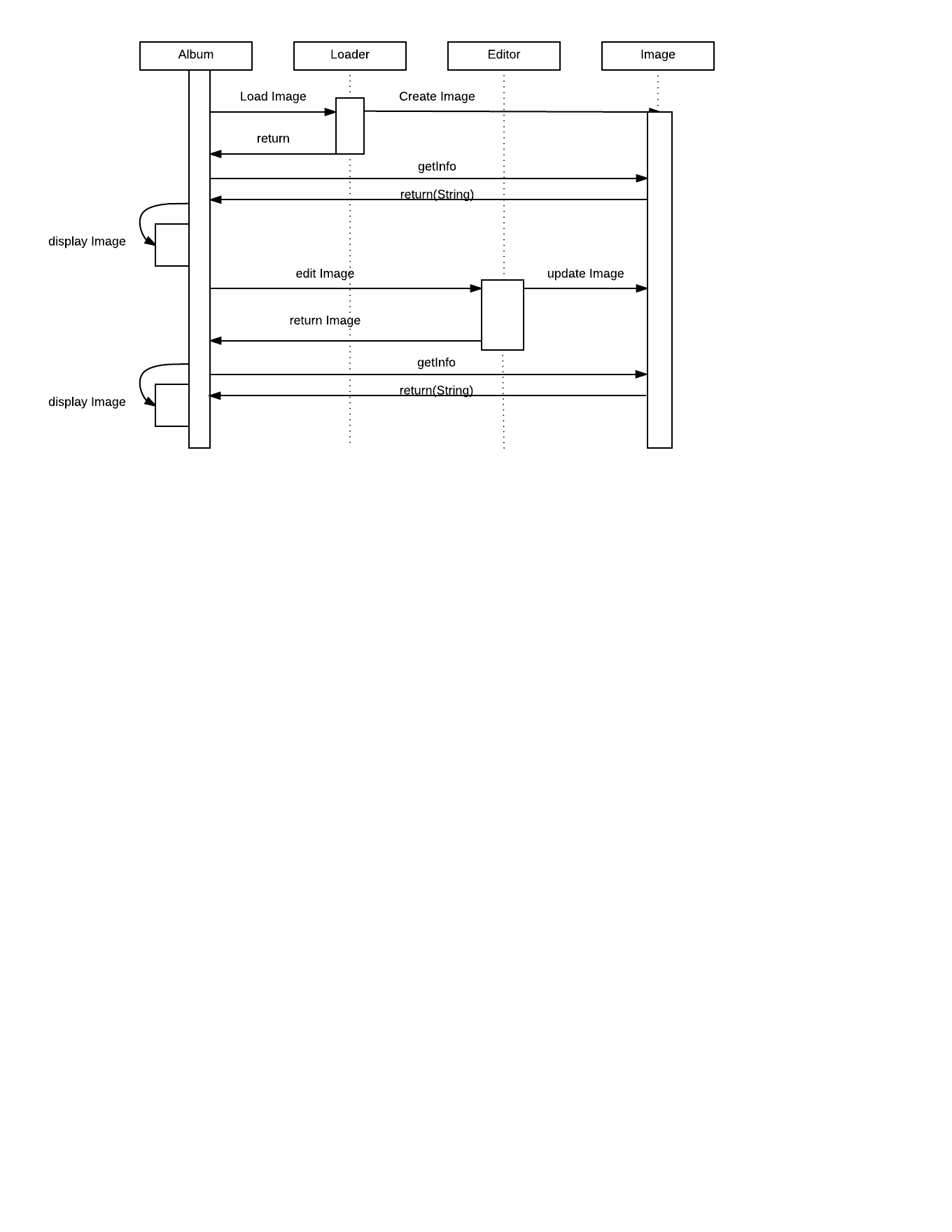


## 

## 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.