Software Design Principles and Patterns

Image Album App
(Image Album Project Part 1)
Documentation version 2

April 5, 2016

Members

Ante Hakstok
Antonio Labriola
Marko Pancirov
Ana Hakstok

Contents

Classes

We have four classes each with a single responsibility in support of the Separation of Concerns principle. The focus is on low coupling and high cohesion.

Album

A collection of Image objects, can add and delete images from that collection. It aggregates Image objects.

Image

Represents the image. It is used to create image objects.
stores data about the image like its size, name, path and labels associated with it. It also has standard getters and setters for these fields.

ImageEditor

Editor class handles image manipulation that is on the front end done by the user. It takes an image object that needs to be edited from the controller class as well as the information about the type of editing(crop, flip, mirror, and rotate images), and afterwards returns the altered image object to the controller. It actually stores a reference of the image being edited.

ImageAlbum

This is a controller class(main method will be in this class) used to run the application. It creates and stores all objects created by using the other classes. It also displays the images for the user to interact with. This class is like a facade for the entire application because the user or other developer can interact with all other classes within this class in a simplified manner.

Libraries

In order to achieve the full functionality that we have intended for our classes we will need to implement JFrame class from the standard java library. We will use this to make an user interface. We will also use utilities such as Java 2D.

Responsibilities and collaborators

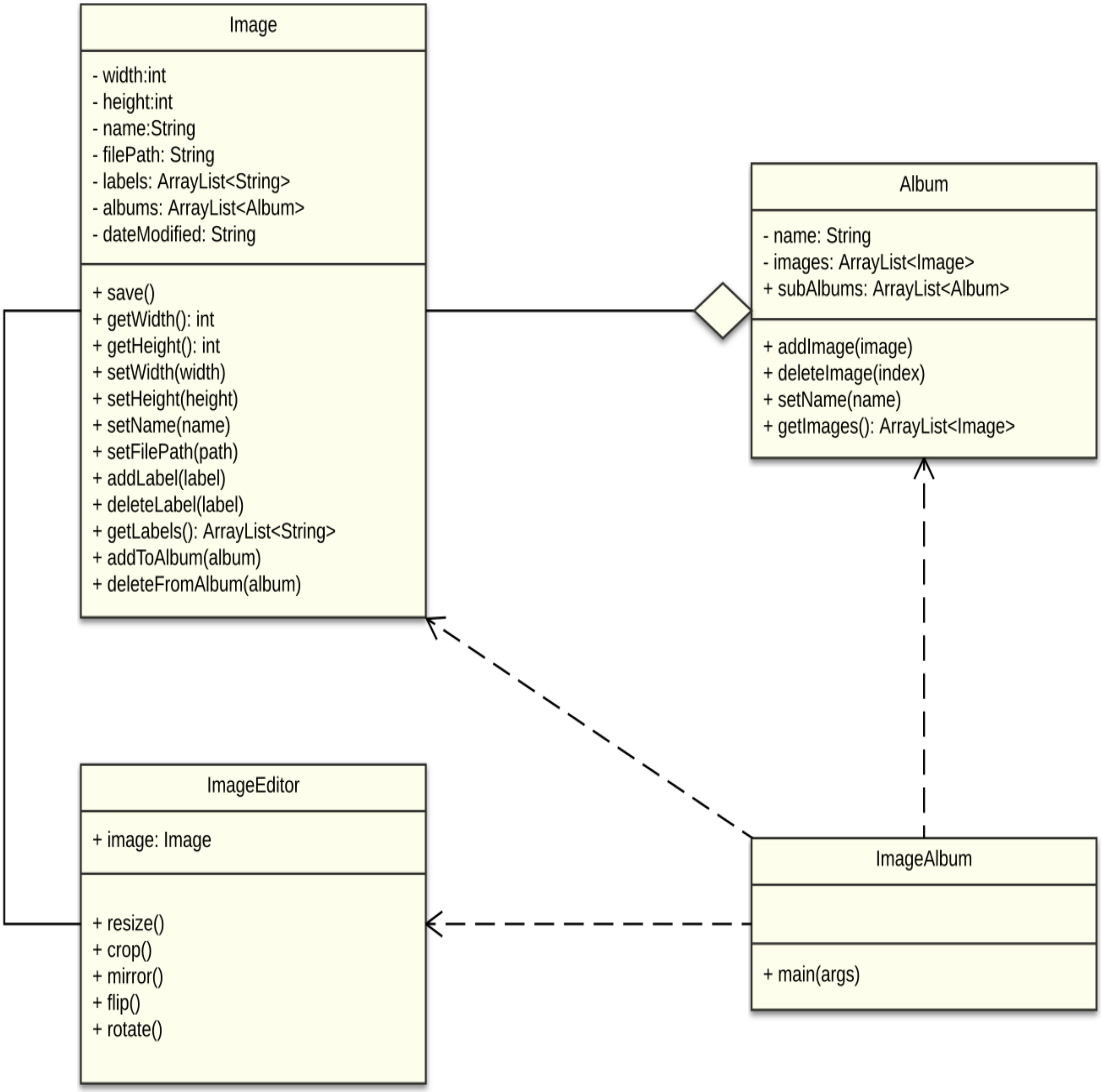
ImageAlbum collaborates with all other classes, it uses **Image** class to create Image objects, **ImageEditor** to manipulate images and **Album** to aggregate them.

ImageEditor retains a reference to the **Image** object that is being edited.

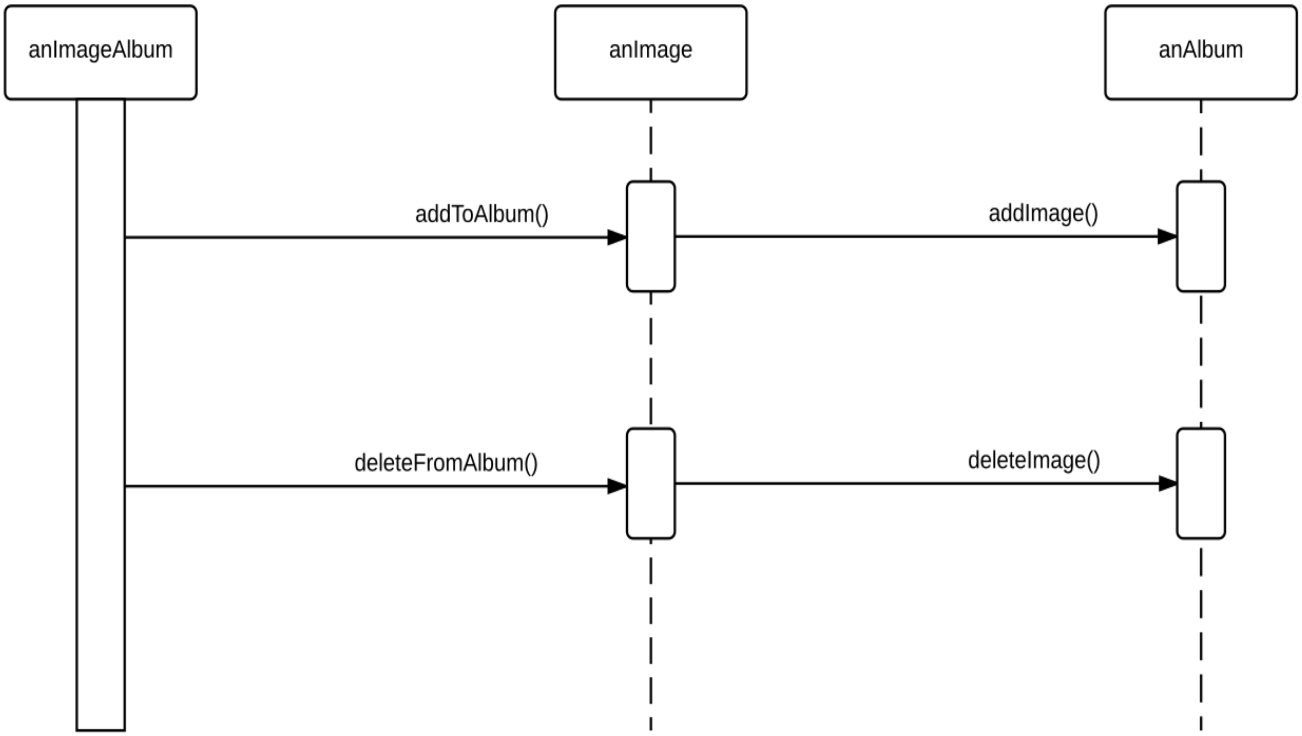
Image class is used by all other classes. **ImageEditor** uses it to manipulate with the image, **Album** aggregates images and **ImageAlbum** creates and handles Image objects.

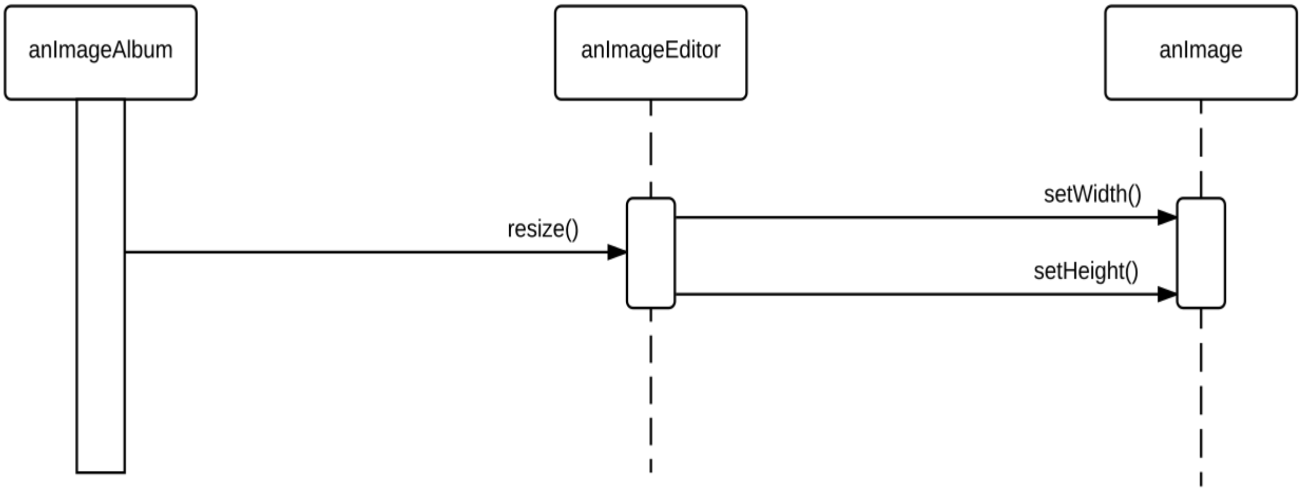
Album is just a collection of **Image** objects.

Class diagram



Sequence Diagram





The following is the material for Image Album Project - Part 2

In order to facilitate multiple image formats and saving image data in multiple formats we used the Builder and Factory Patterns.

Multiple image file formats - Builder Pattern

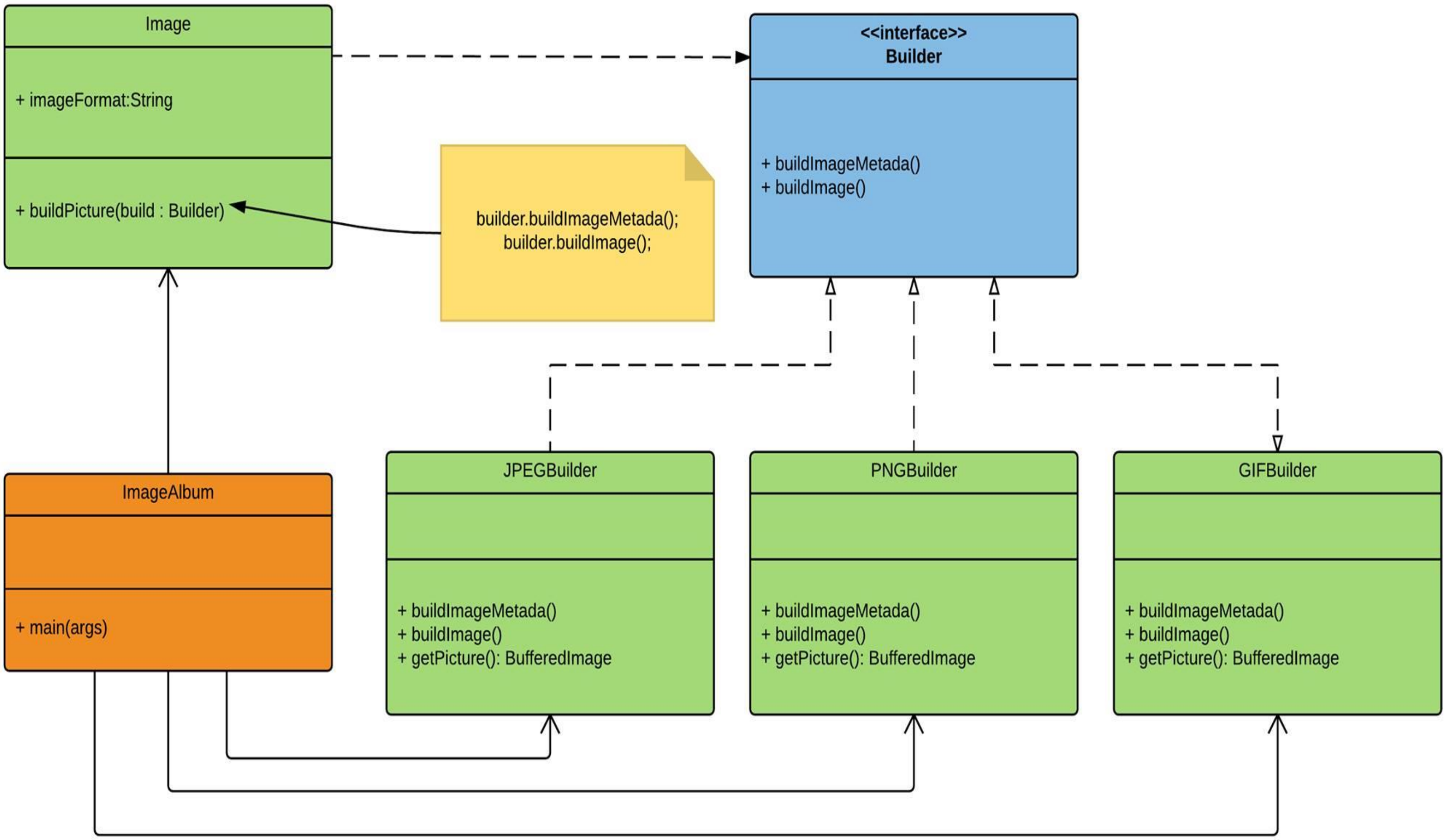
The user can now save the image in three formats: PNF, GIF, JPEG
The Image class uses the buildPicture() method which is called inside of the save() method. This method calls the builder methods.

The ConcreteBuilders are JPEGBuilder, PNGBuilder and GIFBuilder

In regards to the Builder Pattern:

Director: Image
Client: ImageAlbum
Builder interface: Builder
Concrete Builders: JPEGBuilder, PNGBuilder, GIFBuilder

Class diagram - multiple image file formats - Builder Pattern



Saving image data in multiple file formats - Factory Pattern

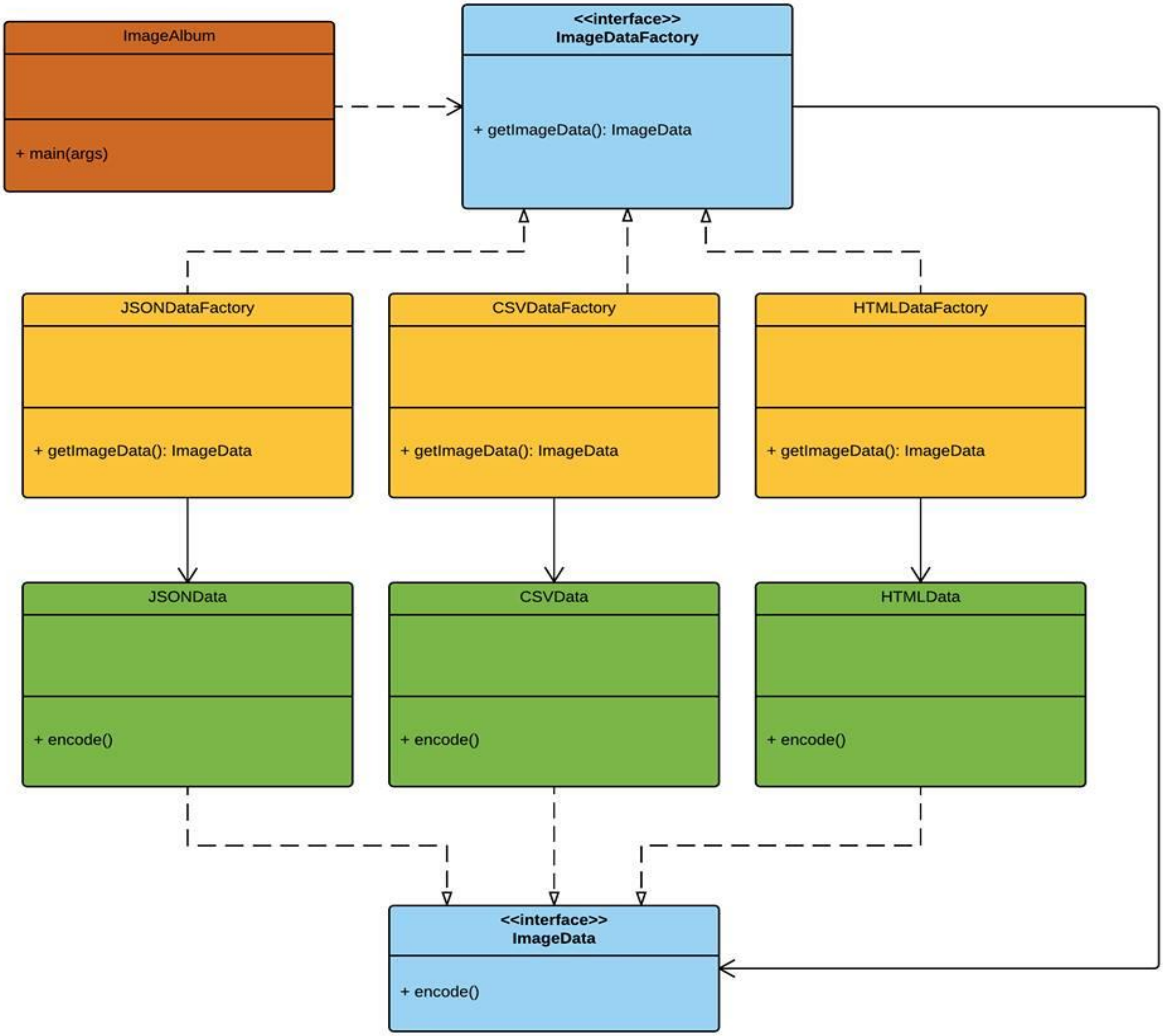
We want to be able to save image data(name, path etc.) in different file formats. In this case we want CSV, JSON and HTML.

ImageAlbum is the client which uses ImageDataFactory to create different file formats for the image data. We have ImageData interface that requires the encode() method to be implemented. This method encodes data in a desired format. For example, JSONData encodes image data in JSON format.

If we want to add a new format, we just need to implement the ImageData and ImageDataFactory interfaces.

On the following page is the class diagram.

Class diagram - multiple image data file formats - Factory Pattern


ImageAlbum
+ main(args)
<<interface>>
ImageDataFactory
+ getImageData(): ImageData
JSONDataFactory
+ getImageData(): ImageData
CSVDataFactory
+ getImageData(): ImageData
HTMLDataFactory
+ getImageData(): ImageData
JSONData
+ encode()
CSVData
+ encode()
HTMLData
+ encode()
<<interface>>
ImageData
+ encode()