System design document for G19

Version: 1.0
Date: 2017-04-05
Author: Anton Hagvall, Niclas Johansson, Hanna Jacobsson, Kristina Markan
This version overrides all previous versions

1 Introduction

1.1 Design goals

Layers is a user-friendly photo-editing software that can be used effectively to make some of the most common changes to photos without the steep learning curve in some other programs.

1.2 Definitions, acronyms and abbreviation

- **Class -** a Java container used as a template for creating objects
- **Filter -** a type of image transformation adjusting pixels slightly
- **GUI -** graphical user interface
- **int** - a integer value containing whole numbers
- **Java** - a platform independent programming language
- **Kernel** - a matrix of values containing weights for a group of pixels
- **Layer stack** - list containing all the layers applied on the image
- **Layer** - saved filters/color/text transformation for easy control and access
- **Object** - a java term used to describe a defined amount of stored data
- **Pixel** - a point of light which contains the component colors of the final color
- **RGB** - Red Green Blue which stands for the component colors of a color which can represent

2 System architecture

2.1 Overview

The program is written in Java using the MVC model.

2.2 Model

The program is built on a model divided into different parts. The main model contains the classes who handles all the functions we have implemented. The controllers are talking to the filter through the class-layers which contain a list of all the layers applied on the image. When it is time to render the image, the controller proceeds through the list and calls each filter's transform method. The transform method takes a loaded image as input and returns a new loaded image with the effects added.

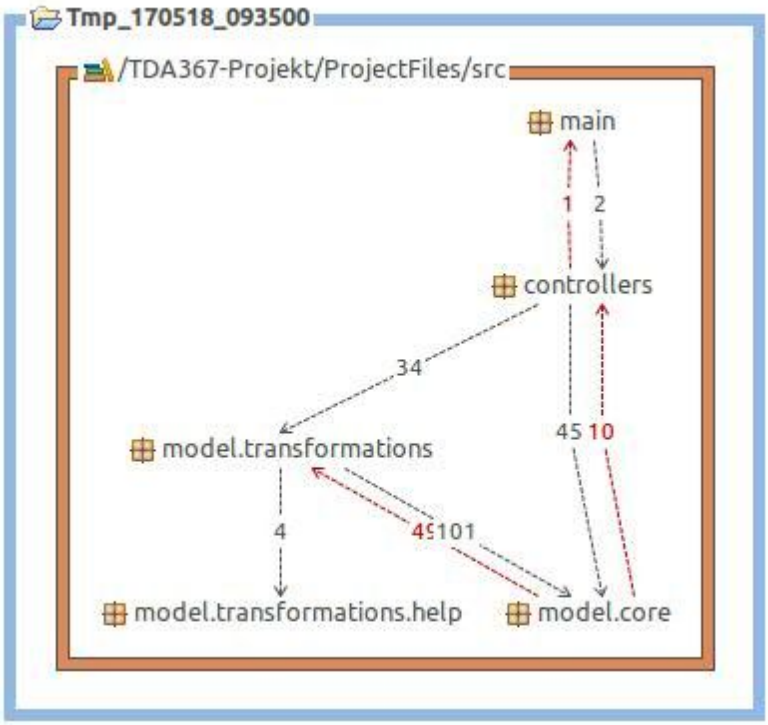
2.3 Layers

In the model, the program have a huge focus on the layer structure.The layer structure has three levels; The highest level is the Layers class storing the layers applied to the image. The second level is the Layer class which contains information of the layer and its visibility.

3 Subsystem decomposition

3.1 Package dependencies

The program follows a straight dependency line. the controller talks to the core in two way. the controllers also has references to all the transformations added when the user adds a new layer.



3.2 Model

3.2.1 Core

The core package is responsible for the core functionality on which all transformations are built on. In the core there are classes responsible for the storing of information on runtime, as well as the logic for saving and opening saved projects and the handlers for the user created filters added though the GUI.

These classes include:

- MainView
- CanvasView
- CropView
- LayerRow

- LayerView
- MiniCanvasView
- NewFilterView
- CreatedFilter
- Layer
- Layerable
- Layers
- LoadedImage
- NewFilterHandeler
- OpenProject
- SaveProject

3.2.2 Transformations

Transformations contain all the different functions that all implements the interface Layerable. These classes all have in common that they are the image transformations the user can apply on the images.

These classes include:

- Contrast
- Crop
- Edge
- Exposure
- GaussianBlur
- Grain
- Grayscale
- HMirroring
- Levels
- NewKernel
- RotateL
- RotateR
- Sharpen
- TextFilter
- VMirroring
- WhiteBalance
- BlackAndWhite
- Blur
- ColorShift

3.2.2.1 Help

Help is responsible for a few classes and methods used by the transformations to do some tasks.

These classes include:

- ColorShiftFactory

- ColorShiftType

3.3 Controllers

The controllers package is responsible for all the visible GUI elements. In the core the main view is responsible for all the buttons and menus. MainView is controlling all the sub-controllers responsible for painting the images and showing the layers.

3.4 Main

Main contains the main method responsible for creating and maintaining the stage.

3.5 Interfaces

3.5.1 Layerable

Layerable is an interface implemented by all the classes that can be put as layers on top of images. The Layerable interface contains 3 methods;

3.5.1.1 Transform

Transform is the method called by the Paint-method on each layer. Transform gets a LoadedImage and returns a new LoadedImage. In Transform, the method applies its own algorithm on the image and does the transformation it is responsible for.

3.5.1.2 SaveLayer

Save layer returns the save string for the current layer. The string contains the name of the layer as well as all the data of the fields stored by the objects. The data is later used by OpenImage to recreate the object at the same state.

3.5.1.3 getName

Returns the name of the layer and are used to display the layer in the layerView.

4 Persistent data management

4.1 Pictures in memory

At runtime the program saves a loaded image as an object of the class LoadedImage. The object is responsible for storing the image in RAM using two formats the programs can use later for transforming the image. The first format is a LoadedImage that stores the image as a two dimensional array containing Color objects. The Color objects represent each pixel in the image and stores the component colors building the image. The format is easy to work with when transformations are done on the image. The second format is a BufferedImage object. The difference between the two is that the pixels are stored as a integer value of the pixel and that this property is faster and easier to use when we render and export the image.

That is why both formats are stored. It is also better on performance because no re-calculations are needed on the image when no transformations are done but this approach is also harder on memory usage.

4.2 Images in exported format

The user can export the images as a png file. When the user does an export the program combines all the applied layers into a final images. The final image is exported and saved at the user's chosen location on the hard drive.

4.3 Images saved as projects

In layers we allow the user to save their work in a project form. These files called .nh files are our own project format which allows the program to save all the layers in their current configuration and then be reloaded into the program on the next startup. The structure of the .nh files are separated in 3 sections;

The first section is responsible for the layers the user added to their image. The layers are structured as a list there all the significant data separated by question marks.

The second section is responsible for the image dimension.

The third section is storing the start image loaded into the program. The image is stored as a grid of pixel values stored as ints. These int are on project loading converted to the RGB-values that we work with in the different filters we have implemented.

As a divider different combinations of question marks are used and read the file line by line to get the expected result.

4.4 Saving user created kernels

The program allows the user to create their own kernels. These kernels are then used in our program to give the image a wide variety of effects. The effects can vary a lot, but they all are based on a 3x3 kernel. These kernels are created in a separate GUI and when the program closes they are stored in .txt files in the program root directory. On startup this file is loaded and turned into a list of created Filter objects stored in newFilterhandeler as a list in memory.

5 Access control and security

NA

6 References