1. Original ImageViewer system.

An original ImageViewer system which we were asked to expand consisted of seventeen classes. The images which we were using in this program were held by a class OFImage, which is a subclass of a BufferedImage class implemented in Java awt package. The reason why the authors of original ImageViewer used their own class for images is that they wanted to have direct access to the pixels by calling getPixel and setPixel methods.

The core of the program was ImageViewer.java class, which constructed the main frame of the program along with components drew on it. Main component of the frame was the ImagePanel. ImagePanel is another extension, this time of the JComponent class. This class is used to present an OFImage currentImage variable, so we can see the picture we load.

Speaking of loading an image, for this purpose, the other authors' own class was used – ImageFileManager. The goal of this class is to load an image from any place in our computer by using system file explorer. This file is then returned again, as an OFImage. It also does the reverse – it can take an OFImage and save it anywhere as a file.

The last interesting feature of the original ImageViewer is a whole family of Filter classes. Filter is an abstract superclass for all image filters in this application. Filters can be applied to OFImages by invoking the apply method. The apply method has different use in each subclass. Now, I will briefly explain the role of each filter subclass.

DarkerFilter – calls the Color class method darker() on every pixel in the image which makes the whole picture a bit darker.

EdgeFilter - An image filter to detect edges and highlight them, a bit like a coloured pencil drawing. It gives every pixel a "smoothed colour" which is the colour value that is the average of this pixel and all the adjacent pixels.

FishEyeFilter - It firstly creates a vertical and horizontal offsets for the pixels to be replaced to create an effect similar to a fisheye camera lens.

GreyScaleFilter – It sets every pixel a colour which is the average of three channels. It makes a picture black and white.

InvertFilter – It set every pixel an inverted colour.

LighterFilter – works the same as DarkerFilter, just calls lighter() method

MirrorFilter – It switches the pixels on the left with the pixel on the right and vice versa to create a mirrored version of an image.

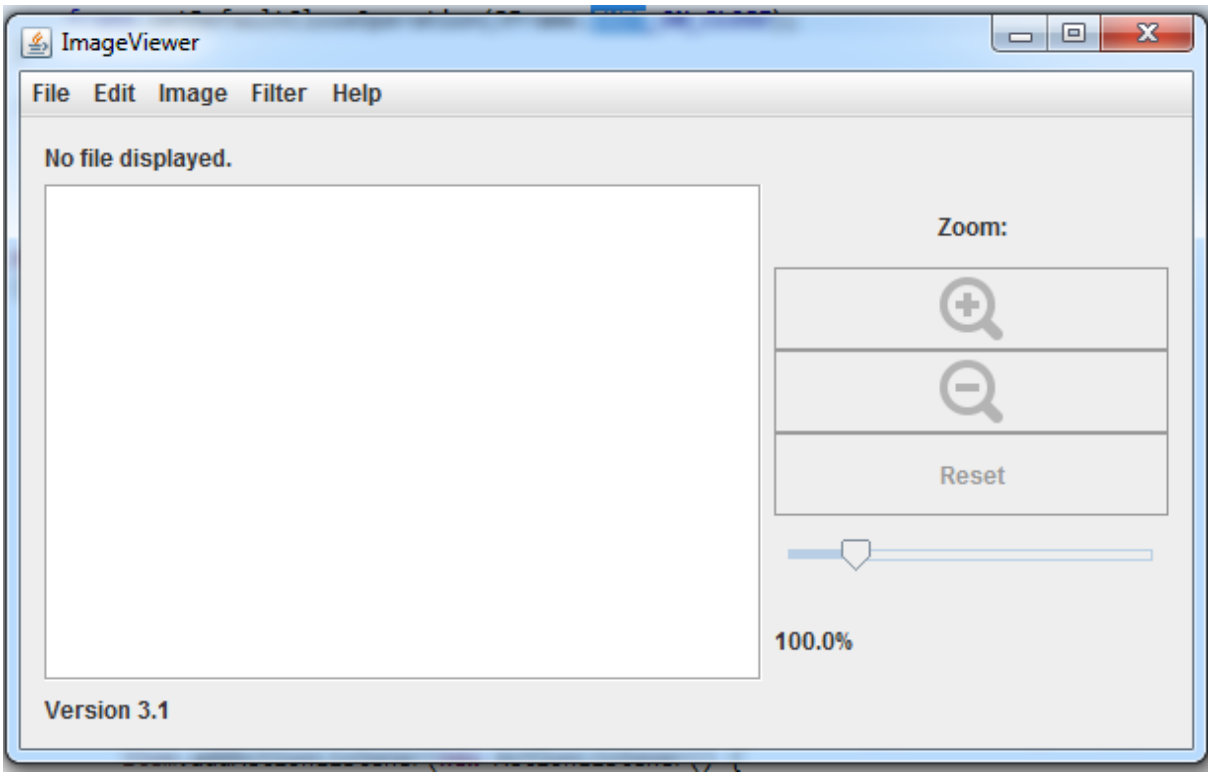
PixelizeFilter – It creates a pixelization effect, like an enlarged low-resolution digital image, the size of each pixel is 5 times the original one.

SmoothFilter – It changes every pixel's colour to "smoothed color" which is the average of this pixel and all the adjacent pixels. It reduces sharp edges and pixelization. A bit like a soft lens.

SolarizeFilter – creates a solarisation effect.

ThresholdFilter - An three-level gray-based threshold filter, based on a brightness of a pixel, it sets it to one of three colour – black, grey or white.

Last class which was used in the original ImageViewer was ImageDriver. It was essentially a class which called a main method, which created an instance of ImageViewer.java class.



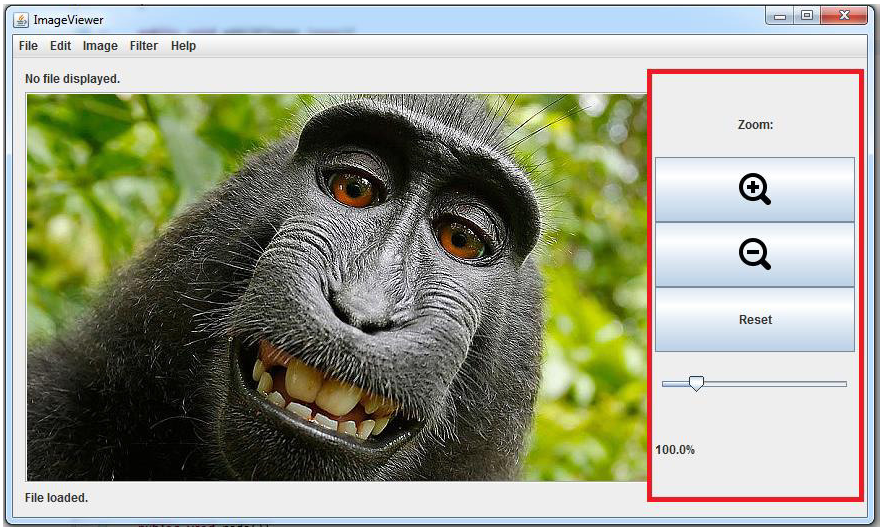
2. Summary of the work I have done

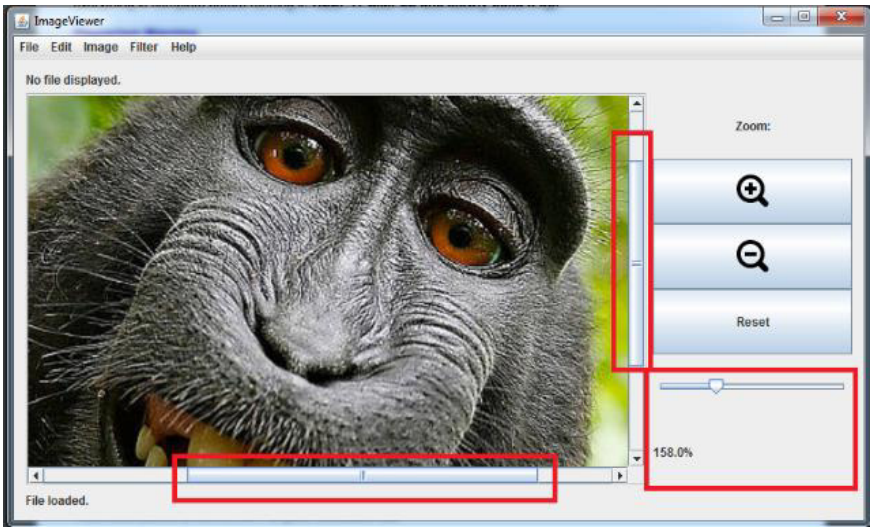
First of all did not like the original loading of an image – a big picture loaded would not fit to my screen. So I created a feature to zoom in and out a picture, so it could have the preferred dimensions of user's choice. I also added scroll panes, so the user could zoom to preffered position of the picture to for example see some specific details of a picture. Then I did not need to have a buttons to make image smaller or bigger, so deleted them. Instead, I created a function to resize the picture itself, as a zooming function was only resizing the picture displayed on ImagePanel, not the picture itself. I created a separate frame for this feature. Then I thought it would be a good idea to implement the similar function to MirrorFilter, which would flip the picture vertically. I created a separate Filter subclass for this feature, FlipVerticallyFilter. The resize and both flipping functions got a separate menu sections called "Image". Next, I came up with the rotate function, as it is common to rotate picture from landscape to portrait perspective. I also added it to Image section of the menu. Lastly, I added HistoryManager class and implemented undo/redo functions candling multiple calls. Throughout the project I also added couple of minor functionalities, not that important from the user's perspective.

3. Detailed explanation of my influence to the code

3.1 Zooming functions and scroll pane

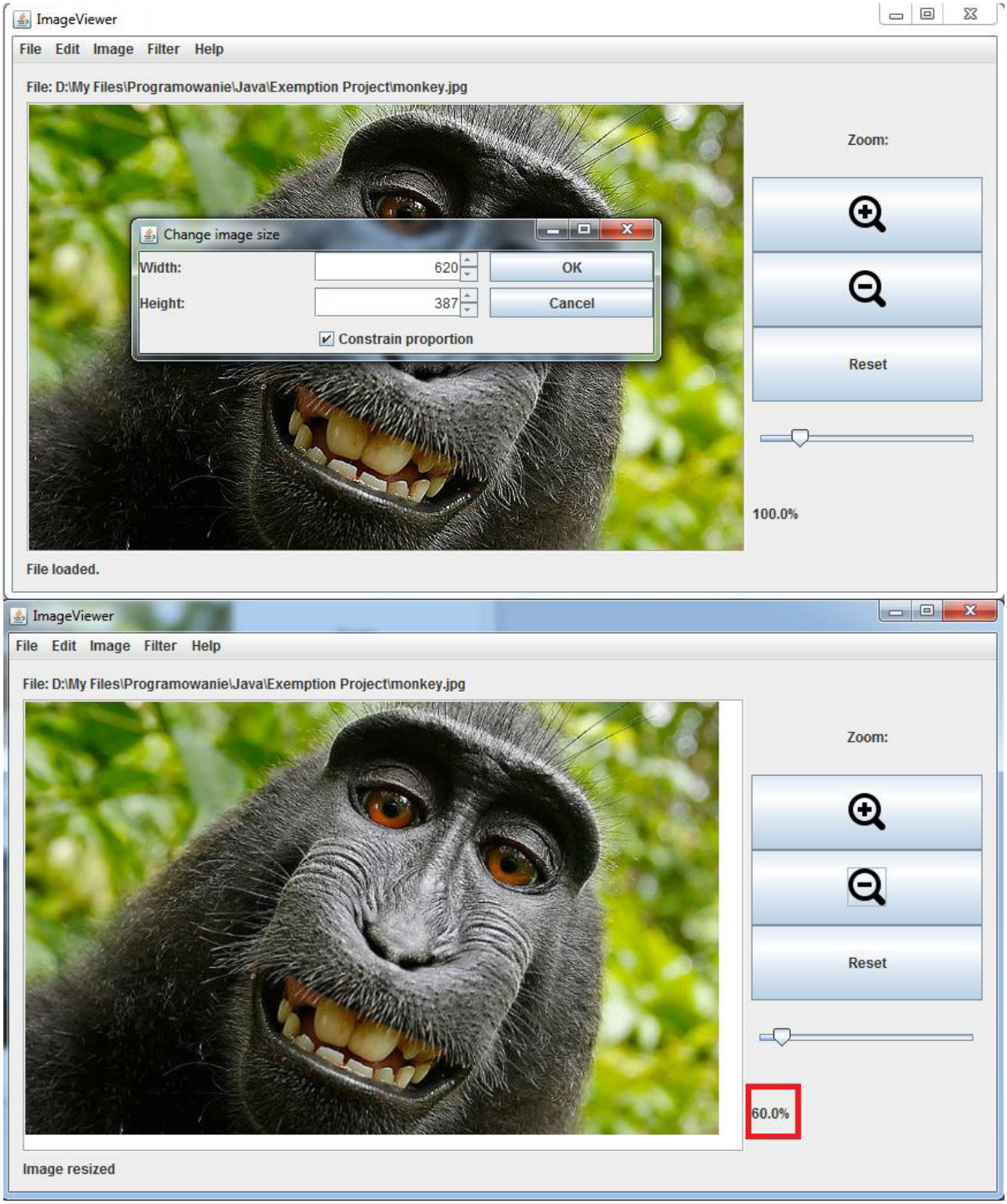
The zoom function is a part of ImagePanel class, as it resizes only the panelImage variable, which is the image shown on the imagePanel. It takes an integer as a parameter, which corresponds to current percentage of zoom (from 50% up to 500% of an original image). It calculates the new dimensions of a panel image and using methods from Graphics2D class it creates a scaled instance of an image[1]. The original panelImage is still kept though, because what I found out it is better to scale from an original image as when we resize back to 100% from let's say 50% we lose quality. Also, I nested the ImagePanel instance of a class in the JScrollPane, to make the image scrollable when it does not fit in the dimensions of a main frame. Now, every time I repaint the frame, zoom or change anything which could cause either main frame or the picture to resize, the following code is being executed: scrollPanel.getViewport().revalidate(); [2] . This is to make sure that scroll panes will refresh and adjust to current situation. The zooming function from the user side is perfomed by either clicking the buttons with the magnifying glasses icons or by using the slider. The label at the bottom keeps the track of the current zoom percentage. When we click the button, the position of a slider also changes, which improves the user's experience of the program. Buttons and sliders are deactivated when there is no image loaded.





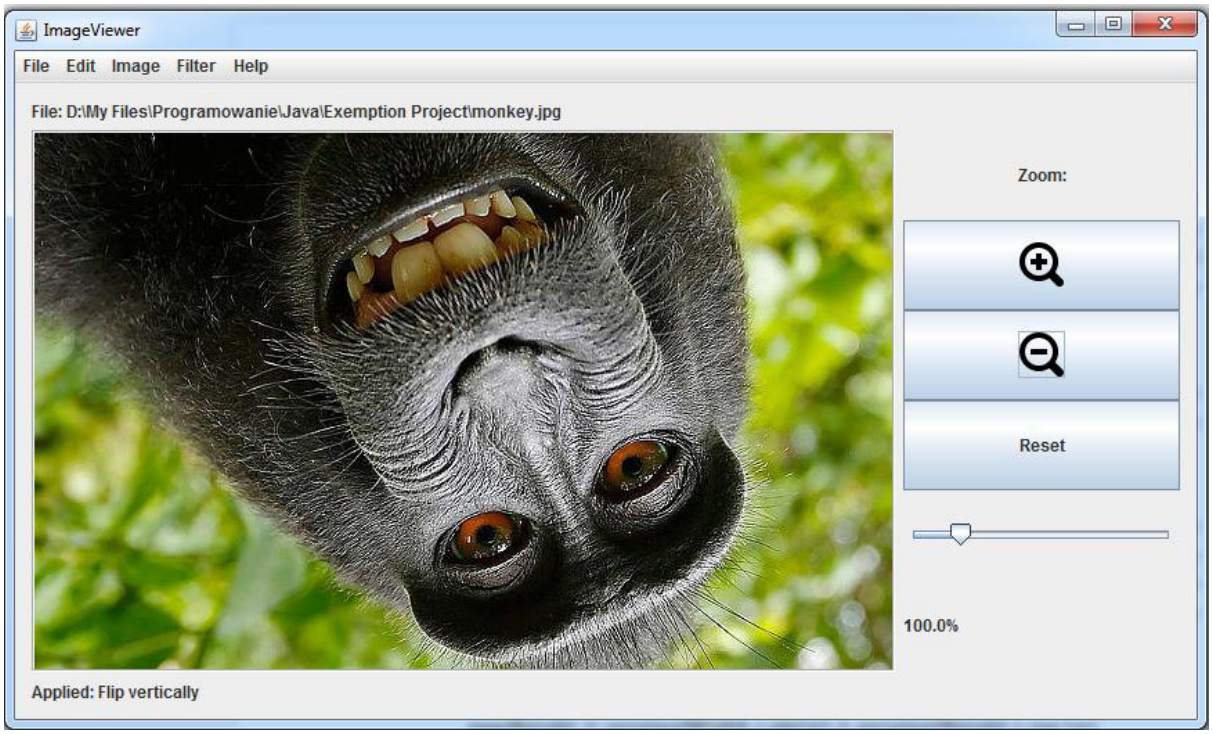
3.2 Resizing the image

For resizing an image I used the same code as for zooming, except it now operates on current image, not the panel image. I created a method, which constructs a new frame with two new components called spinners. I used the type of spinner of number model spinner as it suited my needs. User can also check the tick box if the constraint of proportion of the new dimensions is needed.



3.3 Flip vertically filter class

I also created a new Filter class for flipping the image vertically. The code is basically the same as for MirrorFilter class, I only changed that the for loop gets the right pixels, from top to bottom and vice versa.



3.4 Rotate function [3]

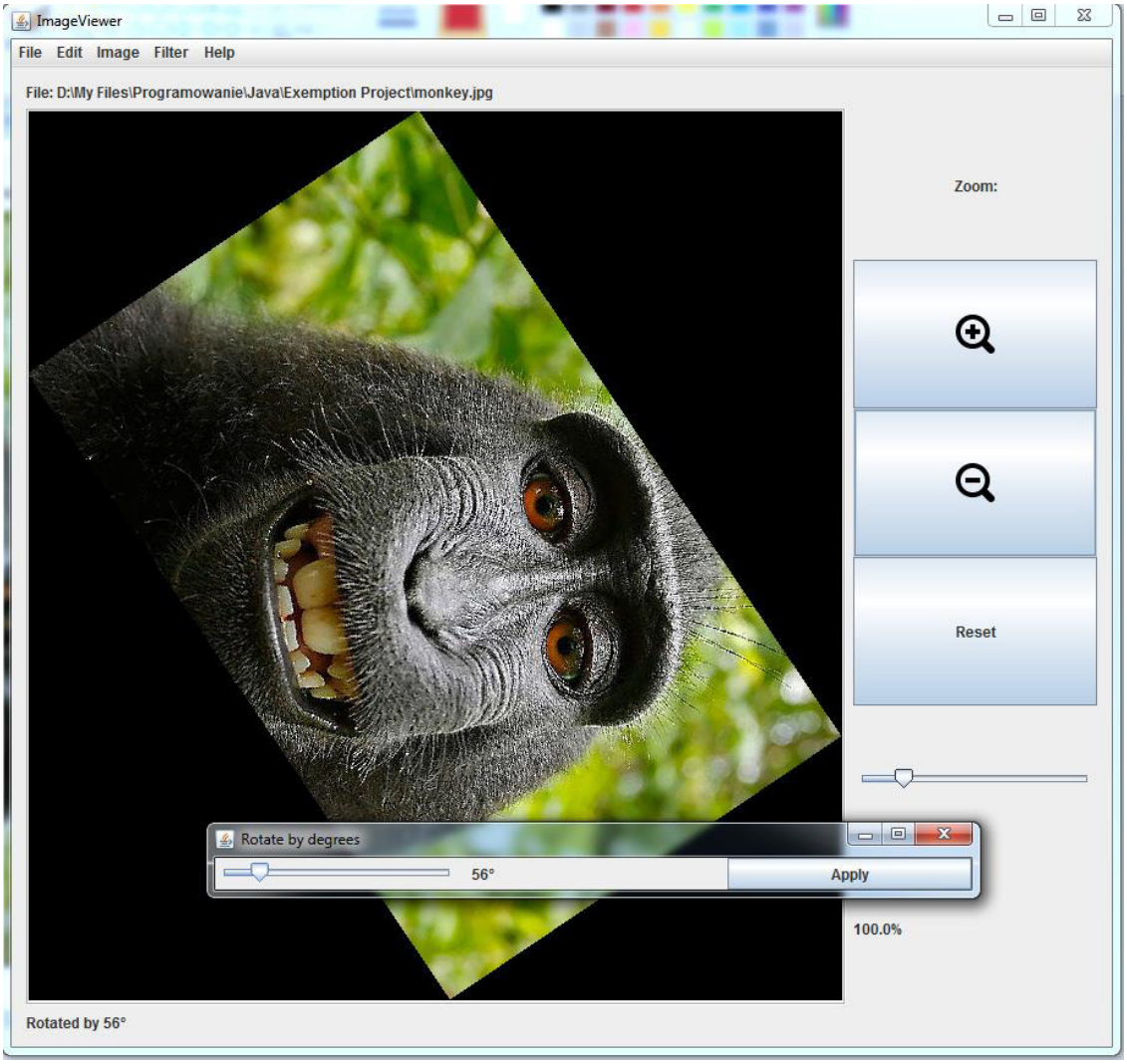
This function takes an angle in degrees for parameter and returns a rotated around the centre instance of an original image. For this function I needed a little revision from the algebra, as my original function was rotating a picture, but unfortunately cutting the parts of image which did not fit in the dimension of an original image. So the function initially calculates the new dimensions of a picture rotated, based on formulas below:

$$newWidth = originalWidth * \cos(\alpha) + originalHeight * \sin(\alpha)$$

$$newHeight = originalWidth * \sin(\alpha) + originalHeight * \cos(\alpha)$$

Then, again using Graphics2D class' methods it is translated to the right place and rotated, so the corners and not lost.

This function is implemented for three different ways in my program: by simply rotating the current image by 90 or 180 degrees or by defining the angle by user input. If the last option is being called, the new frame is drew, with a simple slider, where user can determine the angle between 0 and 360 degrees.



3.5 HistoryManager class, undo/redo function

Lastly, I wanted to implement a functionality of undoing and redoing tasks in my program. For this purpose I created a new class HistoryManager. This class holds two basic things – the array list of images and a current step, which corresponds to the index of array list, which correspond to current image display. Every time we change the something with the current picture displayed the new image is added to an array list and the current step variable is being incremented. When the user undoes couple operation and make a new change to the image, all of the images held in array list above the previous step are being erased and the new image comes at the index of currentStep+1. If the new image is opened or we close the current one, the whole array list is being erased.

3.6 Various other things I added / changed

- openExample() function, it loads a picture of a monkey which is place in main folder of a project. User can call this method with shortcut CTRL+1. Used for quicker testing of the program.
- refreshFrame() function, it does the same as frame.redraw(), used for undo/redo function as I could not reach the frame methods from action listener anonymous inner class
- When opening the image, whole main frame is being centred
- The program closes now after clicking X in the corner, not just hides [4]
- I deleted the unused imported libraries

4. Key lessons from CS105 course I used in the project

- Use of static and local variables
- Developing own classes, creating constructors, methods and fields
- Creating cooperating objects
- Use of for and while loops
- Use of arrays
- Using Eclipse environment
- Inheritance and abstract classes
- Handling exceptions and I/O of files
- Creating GUI
- Casting variables and objects

5. References

1. “charisis” answer to question on StackOverflow: http://stackoverflow.com/questions/4216123/how-to-scale-a-bufferedimage
2. “pete stein” post on coderanch.com forum http://www.coderanch.com/t/497713/GUI/java/JScrollPane-update
3. “Sri Harsha Chilakapati” answer to question on StackOverflow: http://stackoverflow.com/questions/15927014/rotating-an-image-90-degrees-in-java
4. “James Shek” response to question on StackOverflow: http://stackoverflow.com/questions/258099/how-to-close-a-java-swing-application-from-the-code
5. Java 8 API