



PixelWizard

Image Editor

OPERATING SYSTEMS

Project Documentation

Dalca Răzvan Toma (dalca.razvan@gmail.com)

Ghețu Mădălina-Ana (madalina.ghetu@gmail.com)

FILS, 1221B

June 2015

CONTENTS

INTRODUCTION	3
RELATED WORK	4
BASIC CONCEPTS	
Graphics	6
Image	8
JComponent	11
Action Listener	12
Mouse Listener/Mouse Motion Listener	12
OVERVIEW	13
CLASS DIAGRAM	14
IMPLEMENTED METHODS	15
USER GUIDE	19
CONCLUSIONS	23
REFERENCES	23

INTRODUCTION

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. During the first two years of our university experience, we had two semesters for Java study, so it became the programming language that is the most accessible to us.

Also, Java is:

- Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- Platform independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
- Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master.
- Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

All those features make Java a good choice for almost any application that you want to develop, so we thought an image editor could be successfully approached in Java, even if working with Graphic Objects is a little tricky. It was quite a challenge for us, but we also learnt a lot of new things.

One of the characteristics of modern photography is that image editing has become a central part of the process. There are two basic ways that images can be adjusted. Pixel editing works at the pixel level and requires altering the original image. Parametric Image Editing works by saving instruction sets that change the appearance of images without actually changing the original image data.

If you browse the web, you will find a variety of image editors, from the basic online editors, to more complex ones, and to professional software. And we shall not forget the basic Microsoft Paint. Most common editors offer basic functions like drawing on a picture, adding effects and rotating.

PixelWizard was designed to provide editing and drawing functions, everything combined in a simple, friendly interface.

RELATED WORK

In this section we will compare PixelWizard with another two similar softwares, PhotoScape and Microsoft Paint.

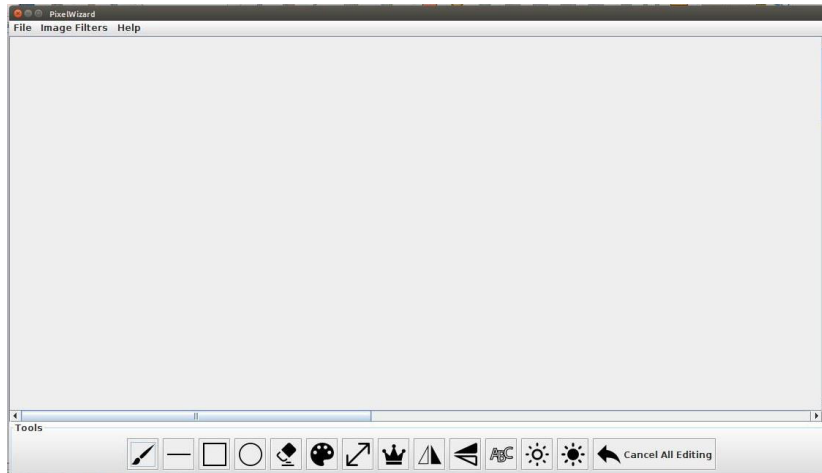


Fig1. - PixelWizard

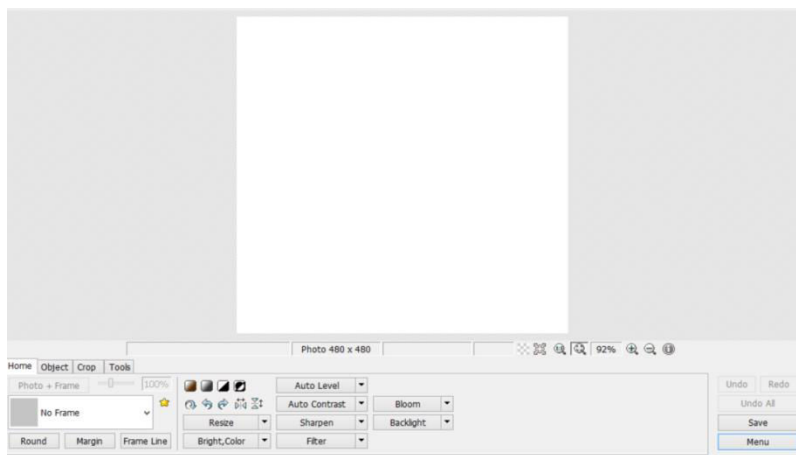


Fig2. - PhotoScape

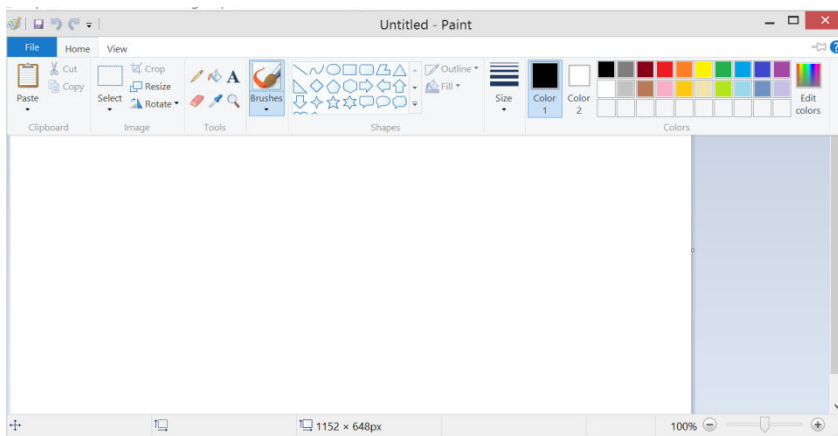


Fig3. - Microsoft Paint

PhotoScape is a fun and easy photo editing software that enables you to fix and enhance photos. (<http://www.photoscape.org/>). It provides basic tools for image editing, such as resizing, brightness and color adjustment, adding text, adding filters, paint brush, just like our program does. In addition to those basic tools, PhotoScape also provides the possibility to merge multiple photos on the page frame to create one final photo, or to take screenshots, which represents a plus that our editor does not implement yet.

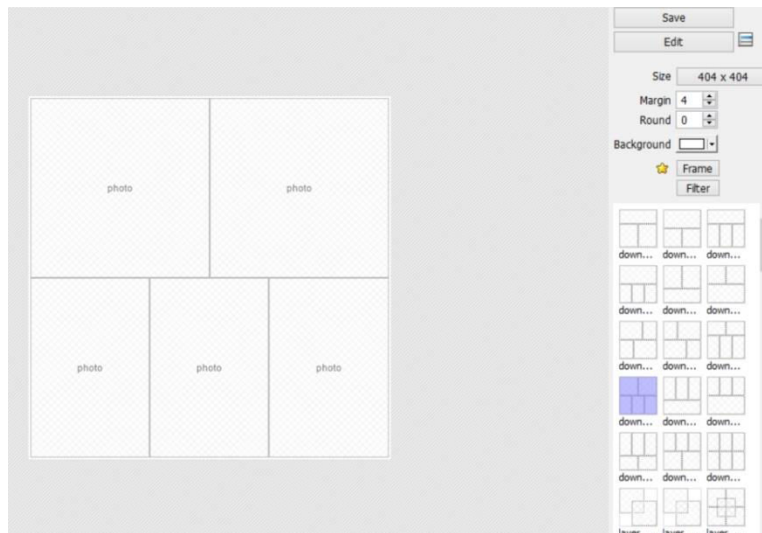


Fig4. - Merge photos in PhotoScape

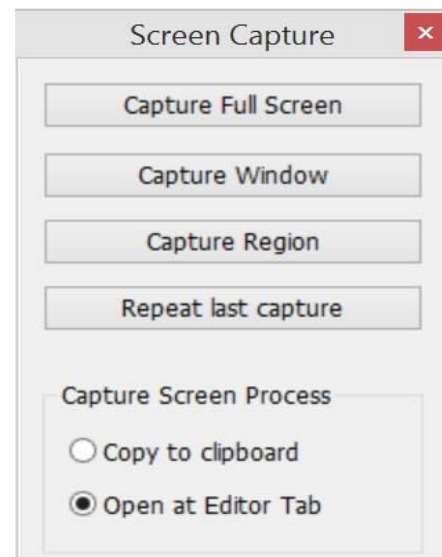


Fig5. - Take screenshot in PhotoScape

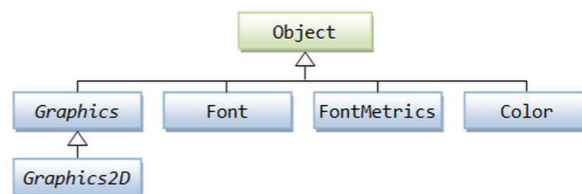
On the other hand, the biggest advantage of PixelEditor is the friendly, intuitive user interface. All the tools are very easy to find and use, as well as the effects, that are all organized in the upper bar, on a drop down list. In PhotoScape some tools may be a little difficult to find or use.

Microsoft Paint is a drawing program you can use to create drawings or edit digital pictures. You can also use Paint to save picture files using different file formats. (<http://windows.microsoft.com/en-us/windows-vista/open-paint>). Both programs are user friendly and offer basic functions. Paint is mostly used for free drawing or shape drawing, having more related tools (more shapes and brush styles). PixelWizard is more complex, allowing the user to draw basic things, but also adding effects to the picture, which is the main advantage over Paint.

BASIC CONCEPTS

1. Graphics

A *graphics context* provides the capabilities of drawing on the screen. The graphics context maintains states such as the color and font used in drawing, as well as interacting with the underlying operating system to perform the drawing. In Java, custom painting is done via the `java.awt.Graphics` class, which manages a graphics context, and provides a set of *device-independent* methods for drawing texts, figures and images on the screen on different platforms.



The `java.awt.Graphics` is an abstract class, as the actual act of drawing is system-dependent and device-dependent. Each operating platform will provide a subclass of `Graphics` to perform the actual drawing under the platform, but conform to the specification defined in `Graphics`.

The methods that are implemented by this class are:

`public abstract boolean drawImage(Image image, int i, int i1, ImageObserver io)`

Parameters:

- img - the specified image to be drawn. This method does nothing if img is null.
- x - the x coordinate.
- y - the y coordinate.
- observer - object to be notified as more of the image is converted.

Returns:

- false if the image pixels are still changing; true otherwise

Draws as much of the specified image as is currently available. The image is drawn with its top-left corner at (x, y) in this graphics context's coordinate space. Transparent pixels in the image do not affect whatever pixels are already there.

This method returns immediately in all cases, even if the complete image has not yet been loaded, and it has not been dithered and converted for the current output device.

If the image has completely loaded and its pixels are no longer being changed, then `drawImage` returns `true`. Otherwise, `drawImage` returns `false` and as more of the image becomes available or it is time to draw another frame of animation, the process that loads the image notifies the specified image observer.

createGraphics()

Creates a `Graphics2D`, which can be used to draw into a `BufferedImage`.

Returns:

a `Graphics2D`, used for drawing into this image.

public abstract void setColor(Color color)

Parameters:

c - the new rendering color.

Sets this graphics context's current color to the specified color. All subsequent graphics operations using this graphics context use this specified color.

public abstract void dispose()

Disposes of this graphics context and releases any system resources that it is using. A `Graphics` object cannot be used after `dispose` has been called.

When a Java program runs, a large number of `Graphics` objects can be created within a short time frame. Although the finalization process of the garbage collector also disposes of the same system resources, it is preferable to manually free the associated resources by calling this method rather than to rely on a finalization process which may not run to completion for a long period of time.

public abstract void setComposite(Composite compst)

Parameters:

comp - the `Composite` object to be used for rendering

Sets the `Composite` for the `Graphics2D` context. The `Composite` is used in all drawing methods such as `drawImage`, `drawString`, `draw`, and `fill`. It specifies how new pixels are

to be combined with the existing pixels on the graphics device during the rendering process.

public abstract void setPaint(Paint paint)

Parameters:

paint - the Paint object to be used to generate color during the rendering process, or null.

Sets the Paint attribute for the Graphics2D context. Calling this method with a null Paint object does not have any effect on the current Paint attribute of this Graphics2D.

public abstract void setFont(Font font)

Parameters:

font - the font.

Sets this graphics context's font to the specified font. All subsequent text operations using this graphics context use this font. A null argument is silently ignored.

public void repaint()

Repaints the component.

If this component is a lightweight component, this method causes a call to this component's paint method as soon as possible. Otherwise, this method causes a call to this component's update method as soon as possible.

2. Image

2.1 The BufferedImage class is a cornerstone of the Java 2D immediate-mode imaging API. It manages the image in memory and provides methods for storing, interpreting, and obtaining pixel data. Since BufferedImage is a subclass of Image it can be rendered by the Graphics and Graphics2D methods that accept an Image parameter.

A BufferedImage is essentially an Image with an accessible data buffer. It is therefore more efficient to work directly with BufferedImage. A BufferedImage has a ColorModel and a Raster of image data. The ColorModel provides a color interpretation of the image's pixel data.

The methods that are implemented by this class are:

public BufferedImage(int i, int i1, int i2)

Parameters:

- width - width of the created image
- height - height of the created image
- imageType - type of the created image

Constructs a BufferedImage of one of the predefined image types. The ColorSpace for the image is the default sRGB space.

public synchronized void setRGB(int i, int i1, int i2)

Parameters:

- x - the X coordinate of the pixel to set
- y - the Y coordinate of the pixel to set
- rgb - the RGB value

Sets a pixel in this BufferedImage to the specified RGB value. The pixel is assumed to be in the default RGB color model, TYPE_INT_ARGB, and default sRGB color space.

public int getRGB(int i, int i1)

Parameters:

- x - the X coordinate of the pixel from which to get the pixel in the default RGB color model and sRGB color space

y - the Y coordinate of the pixel from which to get the pixel in the default RGB color model and sRGB color space.

Returns an integer pixel in the default RGB color model (TYPE_INT_ARGB) and default sRGB colorspace.

public int getHeight()

Returns the height of the image.

public int getWidth()

Returns the width of the image.

public Graphics getGraphics()

Creates a graphics context for drawing to an off-screen image. This method can only be called for off-screen images.

Returns:

a graphics context to draw to the off-screen image.

createGraphics()

Creates a Graphics2D, which can be used to draw into this BufferedImage.

Returns:

a Graphics2D, used for drawing into this image.

2.2 ImageIO is a class containing static convenience methods for locating ImageReaders and ImageWriters, and performing simple encoding and decoding.

public static BufferedImage read(File file) throws IOException

Returns a BufferedImage as the result of decoding a supplied File with an ImageReader chosen automatically from among those currently registered. The File is wrapped in an ImageInputStream. If no registered ImageReader claims to be able to read the resulting stream, null is returned.

public static boolean write(RenderedImage ri, String string, File file)

Parameters:

im - a RenderedImage to be written.

formatName - a String containing the informal name of the format.

output - a File to be written to.

Writes an image using an arbitrary ImageWriter that supports the given format to a File. If there is already a File present, its contents are discarded.

Returns false if no appropriate writer is found.

2.3 public interface BufferedImageOp

This interface describes single-input/single-output operations performed on BufferedImage objects. It is implemented by AffineTransformOp, ConvolveOp, ColorConvertOp, RescaleOp, and LookupOp. These objects can be passed into a BufferedImageFilter to operate on a BufferedImage in the ImageProducer-ImageFilter-ImageConsumer paradigm.

Classes that implement this interface must specify whether or not they allow in-place filtering-- filter operations where the source object is equal to the destination object.

This interface cannot be used to describe more sophisticated operations such as those that take multiple sources. Note that this restriction also means that the values of the destination pixels prior to the operation are not used as input to the filter operation.

2.4 public class Kernel extends [Object](#) implements [Cloneable](#)

The Kernel class defines a matrix that describes how a specified pixel and its surrounding pixels affect the value computed for the pixel's position in the output image of a filtering operation. The X origin and Y origin indicate the kernel matrix element that corresponds to the pixel position for which an output value is being computed.

3. JComponent

The JComponent class extends the Container class, which itself extends Component. The Component class includes everything from providing layout hints to supporting painting and events. The Container class has support for adding components to the container and laying them out.

3.1 JFrame class is used as a top-level container for the graphical components of a graphical user interface (GUI). An instance of JFrame is used to build the primary window of applications. A JFrame can also be used to create secondary windows to an application (or an applet).

3.2 JMenuBar is an implementation of a menu bar. You add JMenu objects to the menu bar to construct a menu. When the user selects a JMenu object, its associated JPopupMenu is displayed, allowing the user to select one of the JMenuItems on it.

3.3 JScrollPane is a specialized container that manages a viewport, optional vertical and horizontal scrollbars, and optional row and column heading viewports.

3.4 The class JButton is an implementation of a push button. This component has a label and generates an event when pressed. It can have Image also.

4. Action Listener

When you click on the JButton, the button fires an ActionEvent. The button registers with some ActionListener before, through the add(ActionListener actListener) method. When an action event is fired, it invokes the actionPerformed (ActionEvent e) method of the ActionListener(s). You don't call the method explicitly but it is implicitly called by computer. An ActionListener is an interface in package java.awt.event. It has only one method: void actionPerformed(ActionEvent e).

5. Mouse Listener/Mouse Motion Listener

Mouse events notify when the user uses the mouse (or similar input device) to interact with a component. Mouse events occur when the cursor enters or exits a component's onscreen area and when the user presses or releases one of the mouse buttons.

Tracking the cursor's motion involves significantly more system overhead than tracking other mouse events. That is why mouse-motion events are separated into Mouse Motion listener type.

METHOD	PURPOSE
void mouseClicked(MouseEvent e)	Called just after the user clicks the listened-to component
void mousePressed(MouseEvent e)	Invoked when a mouse button has been pressed on a component
void mouseReleased(MouseEvent e)	Invoked when a mouse button has been released on a component
void mouseEntered(MouseEvent e)	Invoked when the mouse enters a component
void mouseExited(MouseEvent e)	Invoked when the mouse exits a component
mouseDragged(MouseEvent e)	Called in response to the user moving the mouse while holding a mouse button down

OVERVIEW

The project is divided in 5 classes: ApplicationFrame, DrawArea, FontSelector, ImageEditorTests, Information.

ApplicationFrame class is the one that manages the graphic user interface (GUI). Here are defined all the menu bars, toolboxes, scroll bars and buttons and each elements is given certain properties or is set to do different actions.

In DrawArea class are implemented most of the features of this program. Here are the methods for opening and saving the image, for effects and for mouse actions.

FontSelector is a class that extends JDialog and is made for the window that appears when you want to choose your color and fonts for introducing text. It has methods that allow the user to select any color and fonts, as well as to set the size of the text he wants to add to the image.

The main class of the program is ImageEditorTests in which the main frame of the program is created:

```
public static void main(String[] args)
{
    ApplicationFrame pixelWizard = new ApplicationFrame();
}
```

The information class only creates a window to display details about the developers of the program.

```

classDiagram
    class DrawArea {
        -BufferedImage imageoriginal
        -int pressedX
        -int size
        -int clipArt_size
        -Color colorSelected
        -String pathToFileSave
        -BufferedImage buff2
        -String pathToFilesOpen_string
        +DrawArea()
        -void loadImage()
        +Graphics doDrawing(Graphics g)
        +Graphics paintComponent(Graphics g)
        -String pathToFileOpen()
        +void saveImage()
        +void clear()
        +void grayImage()
        +void negativeImage()
        +void addTextToImage()
        +void brightUp()
        +void brightDown()
        +void sepia()
        +void blur()
        +BufferedImage createHorizontalFlipImage()
        +void horizontalFlip()
        +BufferedImage createVerticalFlipImage()
        +void verticalFlip()
        +BufferedImage createRotatedImage()
        +void rotateImage()
        +void createClipArt()
        +void clipArt()
        +MouseEvent mouseDragged(MouseEvent e)
        +MouseEvent mouseMoved(MouseEvent e)
        +Color chooseColor()
        +int chooseSize()
        +void cancelEditing()
    }

    class ApplicationFrame {
        -JMenu menu1
        -JMenuBar menubar
        -JButton brush
        -JMenuItem newb
        -JToolBar tool_box
        -DrawArea canvas
        -JScrollPane
        -static int counter
        -static int c
        -static FontSelector fs1
        -JButton cancelediting
        +ApplicationFrame()
        +void making_menu()
        +void making_tool_box()
        +ActionEvent actionPerformed(ActionEvent e)
    }

    class JFrame {
        -String title
        -int width
        -int height
        -boolean visible
        +void JFrame()
        +String JFrame(String s)
        +int getHeight()
        +int getWidth()
        +boolean isVisible()
        +int setSize(int w, int h)
        +boolean setVisible(boolean b)
    }

    class JPanel {
    }

    class ImageEditorTests {
        +static String[] main(String[] args)
    }

    class FontSelector {
        -JColorChooser colorChooser
        -JComboBox fontName
        -JTextField fontSize
        -JLabel previewLabel
        -SimpleAttributeSet attributes
        -Font newFont
        -Color newColor
        +FontSelector(Frame parent)
        +ActionEvent actionPerformed(ActionEvent ae)
        +void updatePreviewFont()
        +void updatePreviewColor()
        +String getNewFont()
        +Color getNewColor()
        +int getFontSize()
        +String getTid()
        +void closeAndSave()
        +void closeAndCancel()
    }

    class JDialog {
    }

    class ActionListener {
        <<interface>>
        +ActionEvent actionPerformed(ActionEvent ae)
    }

    class MouseMoutionListener {
        <<interface>>
        +ActionEvent MouseDragged(ActionEvent e)
    }

    class MouseListener {
        <<interface>>
        +MouseEvent mouseClicked(MouseEvent e)
        +MouseEvent mousePressed(MouseEvent e)
        +MouseEvent mouseReleased(MouseEvent e)
    }

    class Information {
        -javax.swing.JButton jButton1
        -javax.swing.JLabel jLabel1
        -javax.swing.JLabel jLabel2
        -javax.swing.JLabel jLabel3
        -javax.swing.JLabel jLabel4
        +Information()
        -// <editor-fold defaultstate="collapsed" desc="Generated Code">
        +GEN-BEGIN: initComponents void initComponents()
        -java.awt.event.ActionEvent jButton1ActionPerformed(java.awt.event.ActionEvent evt)
    }

    DrawArea "1" -- "1" ApplicationFrame : canvas
    ApplicationFrame "1" -- "1" JFrame : is
    ApplicationFrame "1" -- "1" FontSelector : fs1
    ApplicationFrame "1" -- "1" JFrame : info
    ApplicationFrame "1" -- "1" ImageEditorTests : PixelWizard
    FontSelector "1" -- "1" JDialog : is
    FontSelector "1" -- "1" ActionListener : <<implements>>
    FontSelector "1" -- "1" MouseMoutionListener : <<implements>>
    FontSelector "1" -- "1" Information : <<implements>>
    MouseMoutionListener "1" -- "1" MouseListener : <<implements>>
    JFrame "1" -- "1" JPanel : is
    JPanel "1" -- "1" Information : is
  
```

IMPLEMENTED METHODS

ApplicationFrame

```
public class ApplicationFrame
extends javax.swing.JFrame
implements java.awt.event.ActionListener
```

Methods Summary

TYPE	METHOD
void	ApplicationFrame() - constructor
void	actionPerformed(java.awt.event.ActionEvent e)
void	making_menu()
void	making_tool_box()

Inside the constructor method `public ApplicationFrame()`, first is called `super()` which is the default constructor for the `JFrame`, creating a new frame that is initially invisible, so it will have to be set visible by the instruction `this.setVisible(true)`. In the constructor is also set the size of the frame, is initialized the scroll pane, are added the menu bar, toolbox and scroll pane to the frame and are created two methods `making_menu()` and `making_tool_box()` that will be defined outside the constructor.

The method `making_menu()` is a void method (does not return anything) that creates a menu bar and then has some menus added to it, as well as buttons added to each menu. In order for the buttons to perform an action when they are pressed, each one has to have an action listener added to it (ex: `open.addActionListener(this);`)

The method `making_tool_box()` is also a void method that creates a toolbox with buttons. Inside this method a label is set for each button (ex: `brush.setToolTipText("Brush");`) and the responding time for the label to appear after the cursor is put on a specific button is set to 1 ms by the instruction `ToolTipManager.sharedInstance().setInitialDelay(1)`.

Considering that we added an action listener on each button, we also have to define what is that action for each. This is done using the method

```
public void actionPerformed(ActionEvent e) {
```

```
    if (e.getSource().equals(brush)) { this line checks which button is clicked. There is
a variable called counterD and one called c which initially are 0 and then change their
value so that for each button that needs to use the mouse dragged, mouse clicked or
```

mouse released functions, there is no confusion between the implemented methods. Each button has assigned a certain method from DrawArea class.

In this class are also implemented methods for showing the documentation and the information box.

DrawArea

```
public class DrawArea
extends javax.swing.JPanel
implements java.awt.event.MouseMotionListener
```

TYPE	METHOD
void	DrawArea() - constructor
void	<u>blur()</u>
void	<u>brightdown()</u>
void	<u>brightup()</u>
void	<u>clear()</u>
void	<u>clipArt()</u>
java.awt.image.BufferedImage	<u>createhorizontalflipImage()</u>
java.awt.image.BufferedImage	<u>createverticalflipImage()</u>
void	<u>doDrawing</u> (java.awt.Graphics g)
void	<u>grayImage()</u>
void	<u>horizontalflip()</u>
void	void loadImage()
void	<u>negativeImage()</u>
void	<u>paintComponent</u> (java.awt.Graphics g)
String	private String pathttoFile_open()
void	<u>saveImage()</u>
void	<u>sepia()</u>

Inside the constructor method public **DrawArea()**, first is called super() which is the default constructor for the JPanel, then is set the size of the window and a mouse motion listener is added and methods for each counter are implemented.

The method private String **pathttoFile_open()** saves in a variable s the path of the image that the user wants to load.

The method void **loadImage()** uses ImageIO.read(new File(s)); to open the image chosen by the user.

public void **doDrawing(Graphics g)** creates a graphic object and the image is drawn on the panel using the drawImage(image, 0, 0, null) method. The last parameter is the ImageObserver class. It is sometimes used for asynchronous loading. When we do not need asynchronous loading of our images, we can just put null there.

public void **paintComponent(Graphics g)** first calls super(), meaning that it calls the default method of the JPanel. The method is overridden by adding the doDrawing method implemented by before.

public void **saveImage()** the method uses a file chooser to let the user choose where he wants to save the image and the method ImageIO.write.

public void **clear()** this method first checks if there is an image opened, and if not, it creates a new BufferedImage and its corresponding graphic. If there is an image already opened, the method creates a new image with white background, having the same width and height as the original, and replaces it.

public void **grayImage()** this method changes the image pixel by pixel. For each pixel, the value of red, green and blue is taken, and is replaced by a new value, using the formula $\text{new_pixel} = (\text{red_pixel} + \text{green_pixel} + \text{blue_pixel}) / 3$.

public void **negativeImage()** this method changes the image pixel by pixel. For each pixel, the value of red, green and blue is taken, and is replaced by a new value, using the formula $\text{new_pixel} = 255 - \text{initial_pixel}$.

public void **sepia()** this method changes the image pixel by pixel. First, the image is turned into grayscale and then for each pixel, the value of red, green and blue is taken, and is replaced by a new value, using the formulas:

$\text{red} = \text{red} + (20 * 2);$

$\text{green} = \text{green} + 20;$

$\text{blue} = \text{blue} - 25;$ a darker blue pixel gives an increased sepia effect

If any value is less than 0 or greater than 255, it takes the value 0, respectively, 255.

public void blur() this method blurs the image. A blur means an unfocused image. To blur an image, we use a convolution operation. It is a mathematical operation which is also used in edge detection or noise elimination. The blur filter operation replaces each pixel in image with an average of the pixel and its neighbors. Convolutions are per-pixel operations. The same arithmetic is repeated for every pixel in the image. A kernel can be thought of as a two-dimensional grid of numbers that passes over each pixel of an image in sequence, performing calculations along the way. Since images can also be thought of as two-dimensional grids of numbers, applying a kernel to an image can be

visualized as a small grid (the kernel) moving across a substantially larger grid (the image).

public void brightup() / public void brightdown() these two methods change the brightness of the image, by using the class RescaleOp. This class performs a pixel-by-pixel rescaling of the data in the source image by multiplying the sample values for each pixel by a scale factor and then adding an offset. The scaled sample values are clipped to the minimum/maximum representable in the destination image. For a brighter image RescaleOp op = new RescaleOp((float) 1.1, 0, null); and for a darker RescaleOp op = new RescaleOp((float) 0.9, 0, null);

public BufferedImage createhorizontalflipImage() this method redraws the image by interchanging the left and right corners of the original image

public BufferedImage createverticalflipImage() this method redraws the image by interchanging the upper and lower corners of the original image

USER GUIDE

Minimum Requirements: this program works on any operating system, for the program itself is only needed a free space on the disk of 2Mb. However, it requires Java 8, so your PC must also fulfill the following requirements:

Windows

- Windows 8 (Desktop)
- Windows 7
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 (64-bit)
- RAM: 128 MB
- Disk space: 124 MB for JRE; 2 MB for Java Update
- Processor: Minimum Pentium 2 266 MHz processor
- Browsers: Internet Explorer 9 and above, Firefox, Chrome

Mac OS X

- Intel-based Mac running Mac OS X 10.8.3+, 10.9+
- Administrator privileges for installation
- 64-bit browser
- A 64-bit browser (Safari, Firefox, or Chrome for example) is required to run Oracle Java on Mac OS X.

Linux

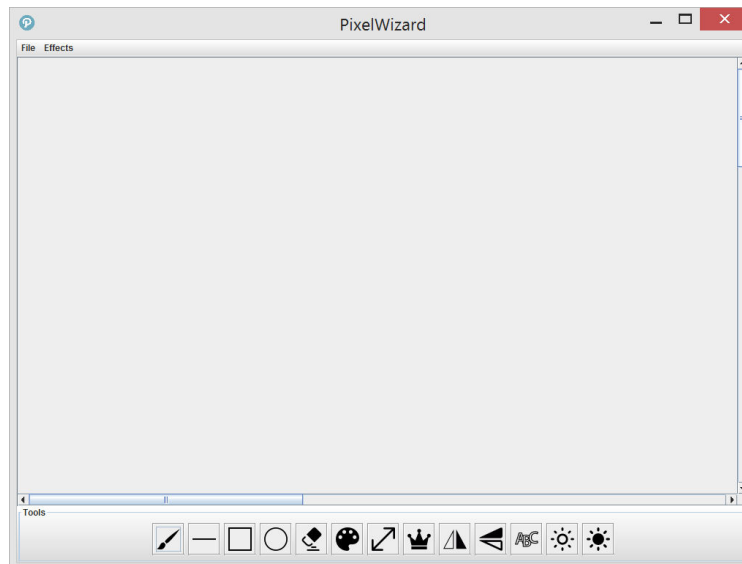
- Oracle Linux 5.5+1
- Oracle Linux 6.x (32-bit), 6.x (64-bit)2
- Oracle Linux 7.x (64-bit)2
- Red Hat Enterprise Linux 5.5+1, 6.x (32-bit), 6.x (64-bit)2
- Ubuntu Linux 12.04 LTS, 13.x
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Browsers: Firefox

If you do not have installed Java 8, you can download it from <http://java.com/en/download/>

Installation:

To install PixelWizard, you only need to copy PixelWizard.jar from the provided CD and copy it to the desired destination (it can be copied anywhere on your PC). Then just double click PixelWizard.jar and your program is ready to use.

PixelWizard is a user-friendly, straightforward program that does not require any advanced computer skills or knowledge in image editing. When you open the program, a window like the one below appears.




First of all, you have to decide if you want just a blank white image to draw onto, or if you want to edit a image that you have on your PC. For a blank image, go to File>New and for a specific image go to File>Open-then a window will open, to let you browse for your desired image. Once you found it, select it and click Open. If you changed your mind and don't want to open an image anymore, click Cancel.


Once you opened your image you have a variety of editing and drawing tools:


Filters - in the upper menu, there is a button called ImageFilters. If you click on it, a list of will appear and you can click on whatever filter you want to add to your image.


Tools - on the lower part of the window, you can find a toolbox that provided different things that can be done for editing your image. If you go with the cursor on a button without clicking it, a label will appear, to let you know what the button does.


Here is a short presentation of the action of each button:


 **BRUSH** - allows you do draw freely on the image, by holding the left mouse button pressed and moving the cursor


 **LINE** - allows you to draw a straight line; to do that you have to press the left mouse button on the spot where you want your line to begin, and by holding the button pressed, to move the cursor to the end point of your line. When you release the button, the line will appear


 **SQUARE** - allows you to draw a square; to do that you have to press the left mouse button on the spot where you want the upper left corner of the square to be, and by holding the button pressed, to move the cursor to where you want the lower right corner of the square to be; basically you move the cursor on the diagonal of the square. When you release the button, the square will appear.


 **CIRCLE** - allows you to draw a square; to do that you have to press the left mouse button on the spot where you want your circle/oval to be and to hold the button and move the cursor on the diameter on the circle. When you release the button, the circle will appear


 **ERASER** - allows you to erase anything from your picture; to do that you have to position the cursor near de area that you want to erase, hold the left button pressed and move the cursor on the area to be erased. After you are done, you can release the button.

 **COLOR PICKER** - allows you to pick a drawing color; after clicking on the button, a window will appear, from where you click on your color and then press OK.

 **LINE THICKNESS** - when you press this button, a window where you have to enter a value will appear. This value represents the thickness of any line or shape that you draw. The recommended values are between 2 and 40.

 **CLIPART** - allows you to add a clipart or another image to your current image. When you click the button, a window will open, to let you browse for your desired clipart/image. Once you found it, select it and click Open. A window where you have to enter a value will appear, representing the size of your clipart. The recommended size for a clipart is around 75. If you changed your mind and don't want to add a clipart anymore, click Cancel. Then you have to click on your image, where you want to put your clipart and it will appear right there.

 **FLIP HORIZONTALLY** - allows you to flip your image horizontally. Just click the button and the image flips.

 **FLIP VERTICALLY** - allows you to flip your image vertically. Just click the button

and the image flips.



TEXT - allows you to add a text your current image. When you click the button, a window will open, to let you insert your text, pick the font, the size and the color. Once you have set the text, click OK. If you changed your mind and don't want to add a text anymore, click Cancel. Then you have to click on your image, where you want to put your text and it will appear right there.



BRIGHTNESS - allows you to make the picture brighter. If you want much more brightness you can click the button several times.



DARKNESS - allows you to make the picture darker. If you want much more darkness you can click the button several times.



CANCEL ALL EDITING - allows you to cancel all the editing that you have done so far.

After you are done with editing and want to save your new image, go to File>Save. A window will appear, where you have to choose the saving location and to give your image a name. After that, press Save. If you have changed your mind and still want to edit your picture to make it even more awesome, press Cancel.

To exit the program, go to File>Close.

To make your life easier, you can also use hotkeys:

Ctrl-N: New

Ctrl-O: Open

Ctrl-S: Save

Ctrl-Esc: Close

Please note that when you go to ImageFilters>Rotate, there is this  sign.

It means that this feature of the program isn't doing the right thing yet and the developers are still working on making it better, as it will be available when the next update is released.

CONCLUSIONS

To summarize, Java can do a lot of things when it comes to image manipulation. The main idea is that it is always needed to have some Graphic objects and Buffered Images. Also, most of the effects are done by changing the image pixel by pixel.

From a personal point of view, we had a very interesting experience while doing this projects. Considering that we weren't well informed about image manipulation in Java or in other programming languages, it was quite hard for us to understand how everything works. But in the end we did and this program is a good start for our future experience with Java.

We learnt how to work with graphic objects and images, but also how to create a GUI from scratch, using code defined by us and not the easy drag and drop.

We are planning to improve this application, to try to make it more professional, starting with some more features to need to be improved or added, such as rotating the image or cropping.

REFERENCES

<http://www.javaworld.com/article/2077282/learn-java/using-the-graphics-class.html>

<http://docs.oracle.com/javase/7/docs/>

<http://java.about.com/od/a/g/Actionlistener.htm>

<http://www.dpbestflow.org/image-editing/image-editing-overview>

https://www3.ntu.edu.sg/home/ehchua/programming/java/J4b_CustomGraphics.html

<http://zetcode.com/gfx/java2d/java2dimages/>

<http://www.javalobby.org/articles/ultimate-image/#2>