CSE 222 HOMEWORK #12

Orhan Aksoy

09104302

TABLE OF CONTENTS

List Of Figures

1. Problem Description

An image clipping program needs to be developed that enables a user to load a gray-scale image file, clip parts of the image, paste clipped images and fill regions.

1.1. Requirements

1.1.1. The user shall be able to load gray-scale image files. The loaded files will be shown on the application window.

1.1.2. The user shall be able to select a part of the image by making consecutive mouse clicks, and finally making a double-click to form an enclosing polygon.

1.1.3. The selected image shall be highlighted by enclosing it with polygon borders.

1.1.4. The user shall be able to fill the selected part of the image with a specific color that he/she picks.

1.1.5. The user shall be able to copy the selected image to the copy buffer.

1.1.6. The user shall be able to paste the image from the copy buffer onto the loaded image.

1.1.7. The user shall be able to save the final image to a file.

2. Analysis

The application will load the image files using the Java libraries. Then, through proper use of MouseListener interface, the mouse clicks will be captured to form the selection polygon. After the final double-click from the user, the enclosing polygon will be finalized.

First, the pixels inside the polygon will be determined. For this purpose, initially, a point inside the polygon is determined. Then, considering it as the first node in a graph, a breadth-first search is run. During the search, every identified node is marked by putting a pixel with a specific color on a secondary image – named *border image*. This image is created at the beginning when the polygon is complete. Then the polygon is drawn onto this image representing the breadth-first limit. During the breadth-first search, every time a pixel is being identified, it is checked by looking at the corresponding pixel's color from the background image. By this way, the limits of the search is determined.

During the breadth-first search, every time a new pixel is identified, this new pixel is stored in a list. This list is then used for both filling and pasting the copied image.

For the purpose of filling, when the user wants to fill the selected image, all of the vertices in the list are traversed, and the corresponding pixel colors are set.

For the purpose of pasting, when the user clicks 'paste' from the menu, selected the image is bound to the mouse location, moving with it. When the user finally clicks the mouse button, the selected image is drawn onto the current image.

2.1. Starting Point Selection

For the breadth-first algorithm to function properly, the starting point must be inside the polygon.

To pick the point, the following algorithm is used:

- Calculate the average 'y' value of the vertices of the polygon.
- Find the minimum and maximum values of x in the vertices of the polygon.
- On the line (min(x), avg(y)) -> (max(x), avg(y)), increment x one by one until a start and an end point is found within the polygon by consecutive point-in-polygon tests. (minDet(x), maxDet(x)).
- Average these two values of x, and return (avgDet(x), avg(y))

2.2. Pixels inside the Polygon

As described briefly above, the pixels inside the polygon are determined by a breadth-first search algorithm.

Before the search, the *border image* is created. The border image initially is a completely white region on which the bounding polygon is drawn with a different color. Then, the search begins by picking a point inside the polygon. This point is added to the pixel queue and the breadth-first loop begins.

Within the loop, one pixel is retrieved from the pixel queue. This pixel is painted to black on the border image, and the corresponding pixel color in the same location in the original image is added to the pixel store, making this node 'finished'. Then, four adjacent pixels are checked from the border image. The ones that are 'white' are added to the pixel queue, and the loop runs again. The loop ends when there's no pixel in the queue.

In the figure below, the image, selected polygon and the border image at an instant during the search is shown.

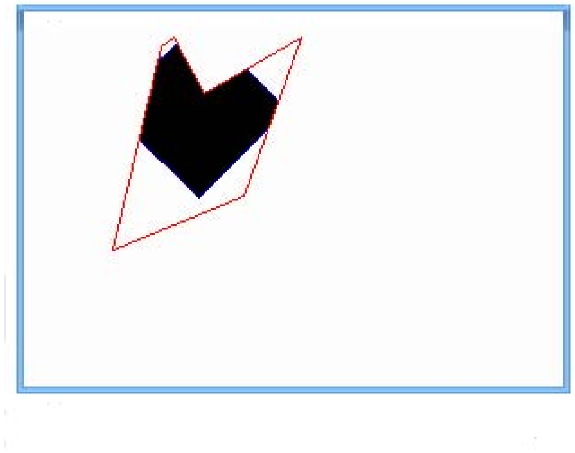
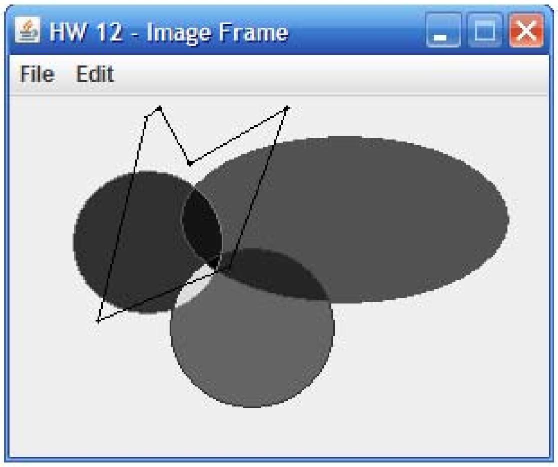

Figure 1 Border Image and the actual Image during the search

2.3. Pasting and Filling

Using the list of pixels that are created during selection, both pasting and filling is trivial.

For filling, the list of pixels is traversed and they are painted on top of the current image. For pasting, the same traversal is run, but with the original pixel colors stored during selection instead of the fill color is used.

3. Design and Implementation

Considering the solution explained in the *Analysis* section, the class diagram is created as show below.

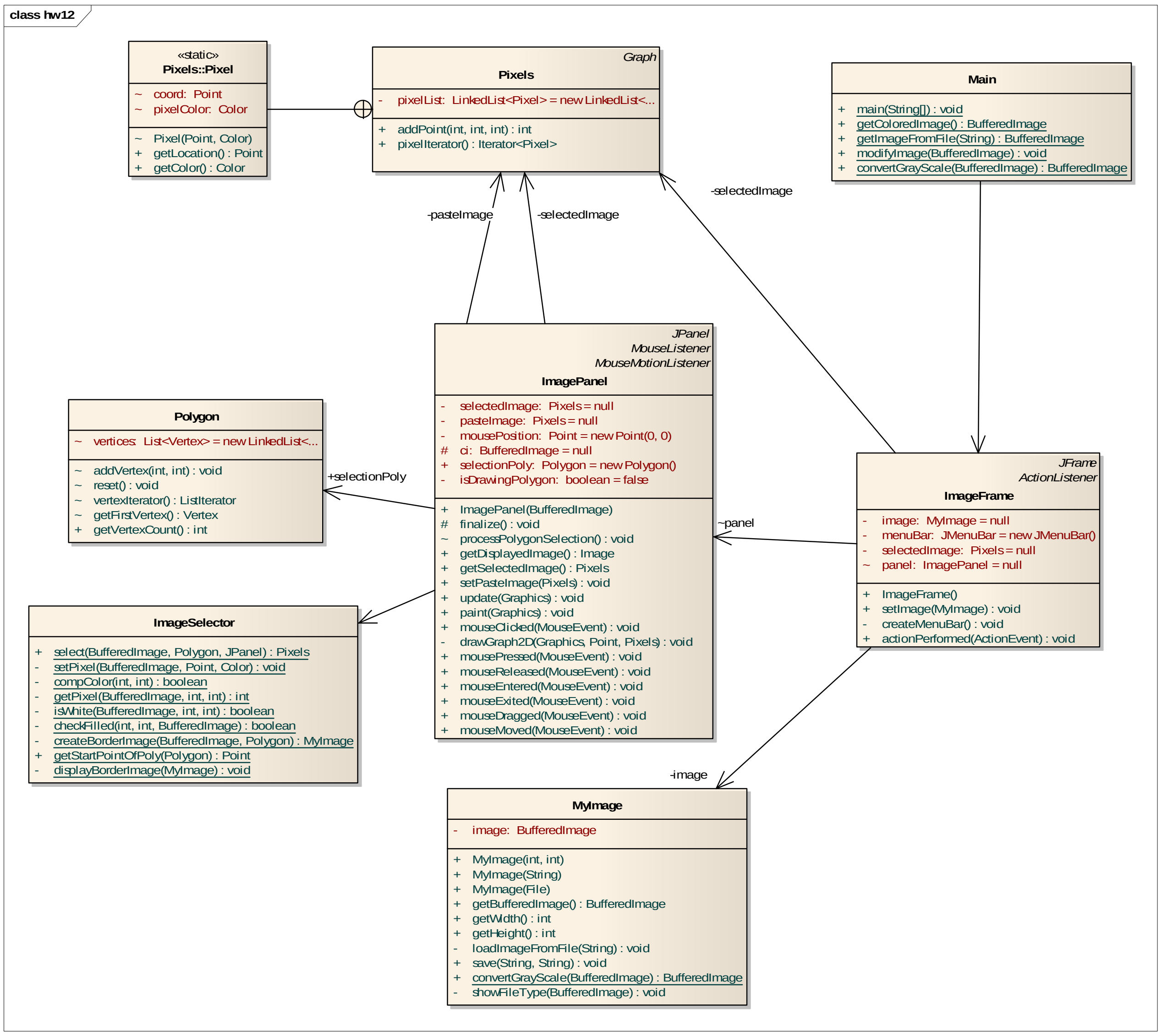
class hw12
«static»
Pixels::Pixel
~ coord: Point
~ pixelColor: Color
~ Pixel(Point, Color)
+ getLocation() : Point
+ getColor() : Color
Graph
Pixels
- pixelList: LinkedList<Pixel> = new LinkedList<...
+ addPoint(int, int, int) : int
+ pixelIterator() : Iterator<Pixel>
Main
+ main(String[]) : void
+ getColoredImage() : BufferedImage
+ getImageFromFile(String) : BufferedImage
+ modifyImage(BufferedImage) : void
+ convertGrayScale(BufferedImage) : BufferedImage
-selectedImage
-pasteImage
-selectedImage
JPanel
MouseListener
MouseMotionListener
ImagePanel
- selectedImage: Pixels = null
- pasteImage: Pixels = null
- mousePosition: Point = new Point(0, 0)
# ci: BufferedImage = null
+ selectionPoly: Polygon = new Polygon()
- isDrawingPolygon: boolean = false
+ ImagePanel(BufferedImage)
# finalize() : void
~ processPolygonSelection() : void
+ getDisplayedImage() : Image
+ getSelectedImage() : Pixels
+ setPasteImage(Pixels) : void
+ update(Graphics) : void
+ paint(Graphics) : void
+ mouseClicked(MouseEvent) : void
- drawGraph2D(Graphics, Point, Pixels) : void
+ mousePressed(MouseEvent) : void
+ mouseReleased(MouseEvent) : void
+ mouseEntered(MouseEvent) : void
+ mouseExited(MouseEvent) : void
+ mouseDragged(MouseEvent) : void
+ mouseMoved(MouseEvent) : void
Polygon
~ vertices: List<Vertex> = new LinkedList<...
~ addVertex(int, int) : void
~ reset() : void
~ vertexIterator() : ListIterator
~ getFirstVertex() : Vertex
+ getVertexCount() : int
+selectionPoly
~panel
JFrame
ActionListener
ImageFrame
- image: MyImage = null
- menuBar: JMenuBar = new JMenuBar()
- selectedImage: Pixels = null
~ panel: ImagePanel = null
+ ImageFrame()
+ setImage(MyImage) : void
- createMenuBar() : void
+ actionPerformed(ActionEvent) : void
ImageSelector
+ select(BufferedImage, Polygon, JPanel) : Pixels
- setPixel(BufferedImage, Point, Color) : void
- compColor(int, int) : boolean
- getPixel(BufferedImage, int, int) : int
- isWhite(BufferedImage, int, int) : boolean
- checkFilled(int, int, BufferedImage) : boolean
- createBorderImage(BufferedImage, Polygon) : MyImage
+ getStartPointOfPoly(Polygon) : Point
- displayBorderImage(MyImage) : void
-image
MyImage
- image: BufferedImage
+ MyImage(int, int)
+ MyImage(String)
+ MyImage(File)
+ getBufferedImage() : BufferedImage
+ getWidth() : int
+ getHeight() : int
- loadImageFromFile(String) : void
+ save(String, String) : void
+ convertGrayScale(BufferedImage) : BufferedImage
- showFileType(BufferedImage) : void

Figure 2 Image Clipper Class Diagram

The class ‘ImageFrame’ extends JFrame, and constitutes the main window of the application. It creates the menu and handles its events, and contains the ‘ImagePanel’ object that is used to paint the image on.

The ImagePanel object not only shows the current image, but also handles the mouse click events during polygon creation, and both click and move events during pasting. After the final double-click event, the polygon is finalized by the ImagePanel object, and the static member ‘select’ of the ImageSelector class is called to create the Pixels object storing the pixels inside the enclosed area.

The MyImage class handles reading/writing image files and providing BufferedImage reference to the users.

4. Testing

In order to test the application, the following sources are necessary:

The sources:

- Main.java, ImageFrame.java, ImagePanel.java, ImageSelector.java, GrayScaleColorChooserPanel.java, MyImage.java, Pixels.java, Polygon.java.
-

The test image:

- test1.bmp, test2.jpg

Before testing, the sources should be compiled using

```
# javac Main.java
```

To run the application, use

```
# java Main
```

4.1. File Load

- On the file menu, select 'Open' and find the file 'test1.bmp, and click 'OK'.
  - Observe that the image is shown on the window.

4.2. Copy & Paste

- Single-click on the window to add the first vertex of the polygon. Then, add new vertices by single-clicks and end with a double click.
  - Observe that the enclosing polgyon is shown on the image.
- Select 'Copy' from the 'Edit' menu.
- On the file menu, select 'Open' and opent the file 'test2.jpg'.
  - test2.jpg is shown on the window.
- Select 'Paste' from the 'Edit' menu.
  - Observe that the copied image moves with the mouse.
- Single click at a location on the image.
  - Observe that the copied image is pasted on that location.

4.3. Fill

- Select a new polygon
  - Observe that the enclosing polgyon is shown on the image.
- Select 'Fill' from the 'Edit' menu
  - Observe that the color selection window opens.
- Select the desired color and accept
  - Observe that the selected polygon is filled with the selected color.

4.4. File Save

- On the file menu, select ‘Save’ and find the file ‘test.bmp, and click ‘OK’.
    - Observe that the image is saved on the specified file.