Name: Anika Jallipalli  
Date: 2/26/2020

14.10 Picture Lab Worksheet

**Directions**: Make note of your responses to the following questions as you work through the activities and exercise in the lesson.

# Activity 1 Questions

1. How many bits does it take to represent the values from 0 to 255?

8 bits

1. How many bytes does it take to represent a color in the RGB color model?

3 bytes

1. How many pixels are in a picture that is 640 pixels wide and 480 pixels high?

307,200 pixels

# Activity 2 Questions

1. How can you make pink?

R = 255 G = 117 B =225

1. How can you make yellow?

R = 255 G = 255 B =0

1. How can you make purple?

R = 255 G = 0 B =225

1. How can you make white?

R = 255 G = 255 B =225

1. How can you make dark gray?

R = 70 G = 70 B =70

# Activity 3 Questions

1. What is the row index for the top left corner of the picture?

0

1. What is the column index for the top left corner of the picture?

0

1. The width of this picture is 640. What is the right-most column index?

639

1. The height of this picture is 480. What is the bottom-most row index?

479

1. Does the row index increase from left to right or top to bottom?

Top to bottom

1. Does the column index increase from left to right or top to bottom?

Left to right

1. Set the zoom to 500%. Describe what you see.   
   This is called pixilation, the picture is magnified so you can see the pixels that look like small squares

# Activity 3 Exercise Results

1. After modifying the main method in the PictureExplorer class to create and explore a different picture from the images folder, paste the code below.

|  |
| --- |
| import java.awt.\*; |
|  | import java.awt.event.\*; |
|  | import javax.swing.\*; |
|  | import java.awt.image.\*; |
|  | import javax.swing.border.\*; |
|  | /\*\* |
|  | \* Displays a picture and lets you explore the picture by displaying the row, column, red, |
|  | \* green, and blue values of the pixel at the cursor when you click a mouse button or |
|  | \* press and hold a mouse button while moving the cursor. It also lets you zoom in or |
|  | \* out. You can also type in a row and column value to see the color at that location. |
|  | \* |
|  | \* Originally created for the Jython Environment for Students (JES). |
|  | \* Modified to work with DrJava by Barbara Ericson |
|  | \* Also modified to show row and columns by Barbara Ericson |
|  | \* |
|  | \* @author Keith McDermottt, gte047w@cc.gatech.edu |
|  | \* @author Barb Ericson ericson@cc.gatech.edu |
|  | \*/ |
|  | public class PictureExplorer implements MouseMotionListener, ActionListener, MouseListener |
|  | { |
|  |  |
|  | // current indicies |
|  | /\*\* row index \*/ |
|  | private int rowIndex = 0; |
|  | /\*\* column index \*/ |
|  | private int colIndex = 0; |
|  |  |
|  | // main GUI |
|  | /\*\* window to hold GUI \*/ |
|  | private JFrame pictureFrame; |
|  | /\*\* window that allows the user to scroll to see a large picture \*/ |
|  | private JScrollPane scrollPane; |
|  |  |
|  | // GUI components |
|  | /\*\* column label \*/ |
|  | private JLabel colLabel; |
|  | /\*\* column previous button \*/ |
|  | private JButton colPrevButton; |
|  | /\*\* row previous button \*/ |
|  | private JButton rowPrevButton; |
|  | /\*\* column next button \*/ |
|  | private JButton colNextButton; |
|  | /\*\* row next button \*/ |
|  | private JButton rowNextButton; |
|  | /\*\* row label \*/ |
|  | private JLabel rowLabel; |
|  | /\*\* text field to show column index \*/ |
|  | private JTextField colValue; |
|  | /\*\* text field to show row index \*/ |
|  | private JTextField rowValue; |
|  | /\*\* red value label \*/ |
|  | private JLabel rValue; |
|  | /\*\* green value label \*/ |
|  | private JLabel gValue; |
|  | /\*\* blue value label \*/ |
|  | private JLabel bValue; |
|  | /\*\* color swatch label \*/ |
|  | private JLabel colorLabel; |
|  | /\*\* panel to show the color swatch \*/ |
|  | private JPanel colorPanel; |
|  | 1. After scaling your image, paste the new main method code below. |
|  | // menu components |
|  | /\*\* menu bar \*/ |
|  | private JMenuBar menuBar; |
|  | /\*\* zoom menu \*/ |
|  | private JMenu zoomMenu; |
|  | /\*\* 25% zoom level \*/ |
|  | private JMenuItem twentyFive; |
|  | /\*\* 50% zoom level \*/ |
|  | private JMenuItem fifty; |
|  | /\*\* 75% zoom level \*/ |
|  | private JMenuItem seventyFive; |
|  | /\*\* 100% zoom level \*/ |
|  | private JMenuItem hundred; |
|  | /\*\* 150% zoom level \*/ |
|  | private JMenuItem hundredFifty; |
|  | /\*\* 200% zoom level \*/ |
|  | private JMenuItem twoHundred; |
|  | /\*\* 500% zoom level \*/ |
|  | private JMenuItem fiveHundred; |
|  |  |
|  | /\*\* The picture being explored \*/ |
|  | private DigitalPicture picture; |
|  |  |
|  | /\*\* The image icon used to display the picture \*/ |
|  | private ImageIcon scrollImageIcon; |
|  |  |
|  | /\*\* The image display \*/ |
|  | private ImageDisplay imageDisplay; |
|  |  |
|  | /\*\* the zoom factor (amount to zoom) \*/ |
|  | private double zoomFactor; |
|  |  |
|  | /\*\* the number system to use, 0 means starting at 0, 1 means starting at 1 \*/ |
|  | private int numberBase=0; |
|  |  |
|  | /\*\* |
|  | \* Public constructor |
|  | \* @param picture the picture to explore |
|  | \*/ |
|  | public PictureExplorer(DigitalPicture picture) |
|  | { |
|  | // set the fields |
|  | this.picture=picture; |
|  | zoomFactor=1; |
|  |  |
|  | // create the window and set things up |
|  | createWindow(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Changes the number system to start at one |
|  | \*/ |
|  | public void changeToBaseOne() |
|  | { |
|  | numberBase=1; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Set the title of the frame |
|  | \*@param title the title to use in the JFrame |
|  | \*/ |
|  | public void setTitle(String title) |
|  | { |
|  | pictureFrame.setTitle(title); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to create and initialize the picture frame |
|  | \*/ |
|  | private void createAndInitPictureFrame() |
|  | { |
|  | pictureFrame = new JFrame(); // create the JFrame |
|  | pictureFrame.setResizable(true); // allow the user to resize it |
|  | pictureFrame.getContentPane().setLayout(new BorderLayout()); // use border layout |
|  | pictureFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE); // when close stop |
|  | pictureFrame.setTitle(picture.getTitle()); |
|  | PictureExplorerFocusTraversalPolicy newPolicy = new PictureExplorerFocusTraversalPolicy(); |
|  | pictureFrame.setFocusTraversalPolicy(newPolicy); |
|  |  |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to create the menu bar, menus, and menu items |
|  | \*/ |
|  | private void setUpMenuBar() |
|  | { |
|  | //create menu |
|  | menuBar = new JMenuBar(); |
|  | zoomMenu = new JMenu("Zoom"); |
|  | twentyFive = new JMenuItem("25%"); |
|  | fifty = new JMenuItem("50%"); |
|  | seventyFive = new JMenuItem("75%"); |
|  | hundred = new JMenuItem("100%"); |
|  | hundred.setEnabled(false); |
|  | hundredFifty = new JMenuItem("150%"); |
|  | twoHundred = new JMenuItem("200%"); |
|  | fiveHundred = new JMenuItem("500%"); |
|  |  |
|  | // add the action listeners |
|  | twentyFive.addActionListener(this); |
|  | fifty.addActionListener(this); |
|  | seventyFive.addActionListener(this); |
|  | hundred.addActionListener(this); |
|  | hundredFifty.addActionListener(this); |
|  | twoHundred.addActionListener(this); |
|  | fiveHundred.addActionListener(this); |
|  |  |
|  | // add the menu items to the menus |
|  | zoomMenu.add(twentyFive); |
|  | zoomMenu.add(fifty); |
|  | zoomMenu.add(seventyFive); |
|  | zoomMenu.add(hundred); |
|  | zoomMenu.add(hundredFifty); |
|  | zoomMenu.add(twoHundred); |
|  | zoomMenu.add(fiveHundred); |
|  | menuBar.add(zoomMenu); |
|  |  |
|  | // set the menu bar to this menu |
|  | pictureFrame.setJMenuBar(menuBar); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Create and initialize the scrolling image |
|  | \*/ |
|  | private void createAndInitScrollingImage() |
|  | { |
|  | scrollPane = new JScrollPane(); |
|  |  |
|  | BufferedImage bimg = picture.getBufferedImage(); |
|  | imageDisplay = new ImageDisplay(bimg); |
|  | imageDisplay.addMouseMotionListener(this); |
|  | imageDisplay.addMouseListener(this); |
|  | imageDisplay.setToolTipText("Click a mouse button on a pixel to see the pixel information"); |
|  | scrollPane.setViewportView(imageDisplay); |
|  | pictureFrame.getContentPane().add(scrollPane, BorderLayout.CENTER); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Creates the JFrame and sets everything up |
|  | \*/ |
|  | private void createWindow() |
|  | { |
|  | // create the picture frame and initialize it |
|  | createAndInitPictureFrame(); |
|  |  |
|  | // set up the menu bar |
|  | setUpMenuBar(); |
|  |  |
|  | //create the information panel |
|  | createInfoPanel(); |
|  |  |
|  | //creates the scrollpane for the picture |
|  | createAndInitScrollingImage(); |
|  |  |
|  | // show the picture in the frame at the size it needs to be |
|  | pictureFrame.pack(); |
|  | pictureFrame.setVisible(true); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to set up the next and previous buttons for the |
|  | \* pixel location information |
|  | \*/ |
|  | private void setUpNextAndPreviousButtons() |
|  | { |
|  | // create the image icons for the buttons |
|  | Icon prevIcon = new ImageIcon(DigitalPicture.class.getResource("leftArrow.gif"), |
|  | "previous index"); |
|  | Icon nextIcon = new ImageIcon(DigitalPicture.class.getResource("rightArrow.gif"), |
|  | "next index"); |
|  | // create the arrow buttons |
|  | colPrevButton = new JButton(prevIcon); |
|  | colNextButton = new JButton(nextIcon); |
|  | rowPrevButton = new JButton(prevIcon); |
|  | rowNextButton = new JButton(nextIcon); |
|  |  |
|  | // set the tool tip text |
|  | colNextButton.setToolTipText("Click to go to the next column value"); |
|  | colPrevButton.setToolTipText("Click to go to the previous column value"); |
|  | rowNextButton.setToolTipText("Click to go to the next row value"); |
|  | rowPrevButton.setToolTipText("Click to go to the previous row value"); |
|  |  |
|  | // set the sizes of the buttons |
|  | int prevWidth = prevIcon.getIconWidth() + 2; |
|  | int nextWidth = nextIcon.getIconWidth() + 2; |
|  | int prevHeight = prevIcon.getIconHeight() + 2; |
|  | int nextHeight = nextIcon.getIconHeight() + 2; |
|  | Dimension prevDimension = new Dimension(prevWidth,prevHeight); |
|  | Dimension nextDimension = new Dimension(nextWidth, nextHeight); |
|  | colPrevButton.setPreferredSize(prevDimension); |
|  | rowPrevButton.setPreferredSize(prevDimension); |
|  | colNextButton.setPreferredSize(nextDimension); |
|  | rowNextButton.setPreferredSize(nextDimension); |
|  |  |
|  | // handle previous column button press |
|  | colPrevButton.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent evt) { |
|  | colIndex--; |
|  | if (colIndex < 0) |
|  | colIndex = 0; |
|  | displayPixelInformation(colIndex,rowIndex); |
|  | } |
|  | }); |
|  |  |
|  | // handle previous row button press |
|  | rowPrevButton.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent evt) { |
|  | rowIndex--; |
|  | if (rowIndex < 0) |
|  | rowIndex = 0; |
|  | displayPixelInformation(colIndex,rowIndex); |
|  | } |
|  | }); |
|  |  |
|  | // handle next column button press |
|  | colNextButton.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent evt) { |
|  | colIndex++; |
|  | if (colIndex >= picture.getWidth()) |
|  | colIndex = picture.getWidth() - 1; |
|  | displayPixelInformation(colIndex,rowIndex); |
|  | } |
|  | }); |
|  |  |
|  | // handle next row button press |
|  | rowNextButton.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent evt) { |
|  | rowIndex++; |
|  | if (rowIndex >= picture.getHeight()) |
|  | rowIndex = picture.getHeight() - 1; |
|  | displayPixelInformation(colIndex,rowIndex); |
|  | } |
|  | }); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Create the pixel location panel |
|  | \* @param labelFont the font for the labels |
|  | \* @return the location panel |
|  | \*/ |
|  | public JPanel createLocationPanel(Font labelFont) { |
|  |  |
|  | // create a location panel |
|  | JPanel locationPanel = new JPanel(); |
|  | locationPanel.setLayout(new FlowLayout()); |
|  | Box hBox = Box.createHorizontalBox(); |
|  |  |
|  | // create the labels |
|  | rowLabel = new JLabel("Row:"); |
|  | colLabel = new JLabel("Column:"); |
|  |  |
|  | // create the text fields |
|  | colValue = new JTextField(Integer.toString(colIndex + numberBase),6); |
|  | colValue.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent e) { |
|  | displayPixelInformation(colValue.getText(),rowValue.getText()); |
|  | } |
|  | }); |
|  | rowValue = new JTextField(Integer.toString(rowIndex + numberBase),6); |
|  | rowValue.addActionListener(new ActionListener() { |
|  | public void actionPerformed(ActionEvent e) { |
|  | displayPixelInformation(colValue.getText(),rowValue.getText()); |
|  | } |
|  | }); |
|  |  |
|  | // set up the next and previous buttons |
|  | setUpNextAndPreviousButtons(); |
|  |  |
|  | // set up the font for the labels |
|  | colLabel.setFont(labelFont); |
|  | rowLabel.setFont(labelFont); |
|  | colValue.setFont(labelFont); |
|  | rowValue.setFont(labelFont); |
|  |  |
|  | // add the items to the vertical box and the box to the panel |
|  | hBox.add(Box.createHorizontalGlue()); |
|  | hBox.add(rowLabel); |
|  | hBox.add(rowPrevButton); |
|  | hBox.add(rowValue); |
|  | hBox.add(rowNextButton); |
|  | hBox.add(Box.createHorizontalStrut(10)); |
|  | hBox.add(colLabel); |
|  | hBox.add(colPrevButton); |
|  | hBox.add(colValue); |
|  | hBox.add(colNextButton); |
|  | locationPanel.add(hBox); |
|  | hBox.add(Box.createHorizontalGlue()); |
|  |  |
|  | return locationPanel; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Create the color information panel |
|  | \* @param labelFont the font to use for labels |
|  | \* @return the color information panel |
|  | \*/ |
|  | private JPanel createColorInfoPanel(Font labelFont) |
|  | { |
|  | // create a color info panel |
|  | JPanel colorInfoPanel = new JPanel(); |
|  | colorInfoPanel.setLayout(new FlowLayout()); |
|  |  |
|  | // get the pixel at the x and y |
|  | Pixel pixel = new Pixel(picture,colIndex,rowIndex); |
|  |  |
|  | // create the labels |
|  | rValue = new JLabel("R: " + pixel.getRed()); |
|  | gValue = new JLabel("G: " + pixel.getGreen()); |
|  | bValue = new JLabel("B: " + pixel.getBlue()); |
|  |  |
|  | // create the sample color panel and label |
|  | colorLabel = new JLabel("Color at location: "); |
|  | colorPanel = new JPanel(); |
|  | colorPanel.setBorder(new LineBorder(Color.black,1)); |
|  |  |
|  | // set the color sample to the pixel color |
|  | colorPanel.setBackground(pixel.getColor()); |
|  |  |
|  | // set the font |
|  | rValue.setFont(labelFont); |
|  | gValue.setFont(labelFont); |
|  | bValue.setFont(labelFont); |
|  | colorLabel.setFont(labelFont); |
|  | colorPanel.setPreferredSize(new Dimension(25,25)); |
|  |  |
|  | // add items to the color information panel |
|  | colorInfoPanel.add(rValue); |
|  | colorInfoPanel.add(gValue); |
|  | colorInfoPanel.add(bValue); |
|  | colorInfoPanel.add(colorLabel); |
|  | colorInfoPanel.add(colorPanel); |
|  |  |
|  | return colorInfoPanel; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Creates the North JPanel with all the pixel location |
|  | \* and color information |
|  | \*/ |
|  | private void createInfoPanel() |
|  | { |
|  | // create the info panel and set the layout |
|  | JPanel infoPanel = new JPanel(); |
|  | infoPanel.setLayout(new BorderLayout()); |
|  |  |
|  | // create the font |
|  | Font largerFont = new Font(infoPanel.getFont().getName(), |
|  | infoPanel.getFont().getStyle(),14); |
|  |  |
|  | // create the pixel location panel |
|  | JPanel locationPanel = createLocationPanel(largerFont); |
|  |  |
|  | // create the color information panel |
|  | JPanel colorInfoPanel = createColorInfoPanel(largerFont); |
|  |  |
|  | // add the panels to the info panel |
|  | infoPanel.add(BorderLayout.NORTH,locationPanel); |
|  | infoPanel.add(BorderLayout.SOUTH,colorInfoPanel); |
|  |  |
|  | // add the info panel |
|  | pictureFrame.getContentPane().add(BorderLayout.NORTH,infoPanel); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to check that the current position is in the viewing area and if |
|  | \* not scroll to center the current position if possible |
|  | \*/ |
|  | public void checkScroll() |
|  | { |
|  | // get the x and y position in pixels |
|  | int xPos = (int) (colIndex \* zoomFactor); |
|  | int yPos = (int) (rowIndex \* zoomFactor); |
|  |  |
|  | // only do this if the image is larger than normal |
|  | if (zoomFactor > 1) { |
|  |  |
|  | // get the rectangle that defines the current view |
|  | JViewport viewport = scrollPane.getViewport(); |
|  | Rectangle rect = viewport.getViewRect(); |
|  | int rectMinX = (int) rect.getX(); |
|  | int rectWidth = (int) rect.getWidth(); |
|  | int rectMaxX = rectMinX + rectWidth - 1; |
|  | int rectMinY = (int) rect.getY(); |
|  | int rectHeight = (int) rect.getHeight(); |
|  | int rectMaxY = rectMinY + rectHeight - 1; |
|  |  |
|  | // get the maximum possible x and y index |
|  | int macolIndexX = (int) (picture.getWidth() \* zoomFactor) - rectWidth - 1; |
|  | int macolIndexY = (int) (picture.getHeight() \* zoomFactor) - rectHeight - 1; |
|  |  |
|  | // calculate how to position the current position in the middle of the viewing |
|  | // area |
|  | int viewX = xPos - (int) (rectWidth / 2); |
|  | int viewY = yPos - (int) (rectHeight / 2); |
|  |  |
|  | // reposition the viewX and viewY if outside allowed values |
|  | if (viewX < 0) |
|  | viewX = 0; |
|  | else if (viewX > macolIndexX) |
|  | viewX = macolIndexX; |
|  | if (viewY < 0) |
|  | viewY = 0; |
|  | else if (viewY > macolIndexY) |
|  | viewY = macolIndexY; |
|  |  |
|  | // move the viewport upper left point |
|  | viewport.scrollRectToVisible(new Rectangle(viewX,viewY,rectWidth,rectHeight)); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Zooms in the on picture by scaling the image. |
|  | \* It is extremely memory intensive. |
|  | \* @param factor the amount to zoom by |
|  | \*/ |
|  | public void zoom(double factor) |
|  | { |
|  | // save the current zoom factor |
|  | zoomFactor = factor; |
|  |  |
|  | // calculate the new width and height and get an image that size |
|  | int width = (int) (picture.getWidth()\*zoomFactor); |
|  | int height = (int) (picture.getHeight()\*zoomFactor); |
|  | BufferedImage bimg = picture.getBufferedImage(); |
|  |  |
|  | // set the scroll image icon to the new image |
|  | imageDisplay.setImage(bimg.getScaledInstance(width, height, Image.SCALE\_DEFAULT)); |
|  | imageDisplay.setCurrentX((int) (colIndex \* zoomFactor)); |
|  | imageDisplay.setCurrentY((int) (rowIndex \* zoomFactor)); |
|  | imageDisplay.revalidate(); |
|  | checkScroll(); // check if need to reposition scroll |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Repaints the image on the scrollpane. |
|  | \*/ |
|  | public void repaint() |
|  | { |
|  | pictureFrame.repaint(); |
|  | } |
|  |  |
|  | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*// |
|  | // Event Listeners // |
|  | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*// |
|  |  |
|  | /\*\* |
|  | \* Called when the mouse is dragged (button held down and moved) |
|  | \* @param e the mouse event |
|  | \*/ |
|  | public void mouseDragged(MouseEvent e) |
|  | { |
|  | displayPixelInformation(e); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to check if the given x and y are in the picture |
|  | \* @param column the horizontal value |
|  | \* @param row the vertical value |
|  | \* @return true if the row and column are in the picture |
|  | \* and false otherwise |
|  | \*/ |
|  | private boolean isLocationInPicture(int column, int row) |
|  | { |
|  | boolean result = false; // the default is false |
|  | if (column >= 0 && column < picture.getWidth() && |
|  | row >= 0 && row < picture.getHeight()) |
|  | result = true; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to display the pixel information from the passed x and y but |
|  | \* also converts x and y from strings |
|  | \* @param xString the x value as a string from the user |
|  | \* @param yString the y value as a string from the user |
|  | \*/ |
|  | public void displayPixelInformation(String xString, String yString) |
|  | { |
|  | int x = -1; |
|  | int y = -1; |
|  | try { |
|  | x = Integer.parseInt(xString); |
|  | x = x - numberBase; |
|  | y = Integer.parseInt(yString); |
|  | y = y - numberBase; |
|  | } catch (Exception ex) { |
|  | } |
|  |  |
|  | if (x >= 0 && y >= 0) { |
|  | displayPixelInformation(x,y); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to display pixel information for the passed x and y |
|  | \* @param pictureX the x value in the picture |
|  | \* @param pictureY the y value in the picture |
|  | \*/ |
|  | private void displayPixelInformation(int pictureX, int pictureY) |
|  | { |
|  | // check that this x and y are in range |
|  | if (isLocationInPicture(pictureX, pictureY)) |
|  | { |
|  | // save the current x and y index |
|  | colIndex = pictureX; |
|  | rowIndex = pictureY; |
|  |  |
|  | // get the pixel at the x and y |
|  | Pixel pixel = new Pixel(picture,colIndex,rowIndex); |
|  |  |
|  | // set the values based on the pixel |
|  | colValue.setText(Integer.toString(colIndex + numberBase)); |
|  | rowValue.setText(Integer.toString(rowIndex + numberBase)); |
|  | rValue.setText("R: " + pixel.getRed()); |
|  | gValue.setText("G: " + pixel.getGreen()); |
|  | bValue.setText("B: " + pixel.getBlue()); |
|  | colorPanel.setBackground(new Color(pixel.getRed(), pixel.getGreen(), pixel.getBlue())); |
|  |  |
|  | } |
|  | else |
|  | { |
|  | clearInformation(); |
|  | } |
|  |  |
|  | // notify the image display of the current x and y |
|  | imageDisplay.setCurrentX((int) (colIndex \* zoomFactor)); |
|  | imageDisplay.setCurrentY((int) (rowIndex \* zoomFactor)); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to display pixel information based on a mouse event |
|  | \* @param e a mouse event |
|  | \*/ |
|  | private void displayPixelInformation(MouseEvent e) |
|  | { |
|  |  |
|  | // get the cursor x and y |
|  | int cursorX = e.getX(); |
|  | int cursorY = e.getY(); |
|  |  |
|  | // get the x and y in the original (not scaled image) |
|  | int pictureX = (int) (cursorX / zoomFactor + numberBase); |
|  | int pictureY = (int) (cursorY / zoomFactor + numberBase); |
|  |  |
|  | // display the information for this x and y |
|  | displayPixelInformation(pictureX,pictureY); |
|  |  |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to clear the labels and current color and reset the |
|  | \* current index to -1 |
|  | \*/ |
|  | private void clearInformation() |
|  | { |
|  | colValue.setText("N/A"); |
|  | rowValue.setText("N/A"); |
|  | rValue.setText("R: N/A"); |
|  | gValue.setText("G: N/A"); |
|  | bValue.setText("B: N/A"); |
|  | colorPanel.setBackground(Color.black); |
|  | colIndex = -1; |
|  | rowIndex = -1; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method called when the mouse is moved with no buttons down |
|  | \* @param e the mouse event |
|  | \*/ |
|  | public void mouseMoved(MouseEvent e) |
|  | {} |
|  |  |
|  | /\*\* |
|  | \* Method called when the mouse is clicked |
|  | \* @param e the mouse event |
|  | \*/ |
|  | public void mouseClicked(MouseEvent e) |
|  | { |
|  | displayPixelInformation(e); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method called when the mouse button is pushed down |
|  | \* @param e the mouse event |
|  | \*/ |
|  | public void mousePressed(MouseEvent e) |
|  | { |
|  | displayPixelInformation(e); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method called when the mouse button is released |
|  | \* @param e the mouse event |
|  | \*/ |
|  | public void mouseReleased(MouseEvent e) |
|  | { |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method called when the component is entered (mouse moves over it) |
|  | \* @param e the mouse event |
|  | \*/ |
|  | public void mouseEntered(MouseEvent e) |
|  | { |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method called when the mouse moves over the component |
|  | \* @param e the mouse event |
|  | \*/ |
|  | public void mouseExited(MouseEvent e) |
|  | { |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to enable all menu commands |
|  | \*/ |
|  | private void enableZoomItems() |
|  | { |
|  | twentyFive.setEnabled(true); |
|  | fifty.setEnabled(true); |
|  | seventyFive.setEnabled(true); |
|  | hundred.setEnabled(true); |
|  | hundredFifty.setEnabled(true); |
|  | twoHundred.setEnabled(true); |
|  | fiveHundred.setEnabled(true); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Controls the zoom menu bar |
|  | \* |
|  | \* @param a the ActionEvent |
|  | \*/ |
|  | public void actionPerformed(ActionEvent a) |
|  | { |
|  |  |
|  | if(a.getActionCommand().equals("Update")) |
|  | { |
|  | this.repaint(); |
|  | } |
|  |  |
|  | if(a.getActionCommand().equals("25%")) |
|  | { |
|  | this.zoom(.25); |
|  | enableZoomItems(); |
|  | twentyFive.setEnabled(false); |
|  | } |
|  |  |
|  | if(a.getActionCommand().equals("50%")) |
|  | { |
|  | this.zoom(.50); |
|  | enableZoomItems(); |
|  | fifty.setEnabled(false); |
|  | } |
|  |  |
|  | if(a.getActionCommand().equals("75%")) |
|  | { |
|  | this.zoom(.75); |
|  | enableZoomItems(); |
|  | seventyFive.setEnabled(false); |
|  | } |
|  |  |
|  | if(a.getActionCommand().equals("100%")) |
|  | { |
|  | this.zoom(1.0); |
|  | enableZoomItems(); |
|  | hundred.setEnabled(false); |
|  | } |
|  |  |
|  | if(a.getActionCommand().equals("150%")) |
|  | { |
|  | this.zoom(1.5); |
|  | enableZoomItems(); |
|  | hundredFifty.setEnabled(false); |
|  | } |
|  |  |
|  | if(a.getActionCommand().equals("200%")) |
|  | { |
|  | this.zoom(2.0); |
|  | enableZoomItems(); |
|  | twoHundred.setEnabled(false); |
|  | } |
|  |  |
|  | if(a.getActionCommand().equals("500%")) |
|  | { |
|  | this.zoom(5.0); |
|  | enableZoomItems(); |
|  | fiveHundred.setEnabled(false); |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Class for establishing the focus for the textfields |
|  | \*/ |
|  | private class PictureExplorerFocusTraversalPolicy |
|  | extends FocusTraversalPolicy { |
|  |  |
|  | /\*\* |
|  | \* Method to get the next component for focus |
|  | \*/ |
|  | public Component getComponentAfter(Container focusCycleRoot, |
|  | Component aComponent) { |
|  | if (aComponent.equals(colValue)) |
|  | return rowValue; |
|  | else |
|  | return colValue; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Method to get the previous component for focus |
|  | \*/ |
|  | public Component getComponentBefore(Container focusCycleRoot, |
|  | Component aComponent) { |
|  | if (aComponent.equals(colValue)) |
|  | return rowValue; |
|  | else |
|  | return colValue; |
|  | } |
|  |  |
|  | public Component getDefaultComponent(Container focusCycleRoot) { |
|  | return colValue; |
|  | } |
|  |  |
|  | public Component getLastComponent(Container focusCycleRoot) { |
|  | return rowValue; |
|  | } |
|  |  |
|  | public Component getFirstComponent(Container focusCycleRoot) { |
|  | return colValue; |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Test Main. It will explore the beach |
|  | \*/ |
|  | public static void main( String args[]) |
|  | { |
|  | Picture pix = new Picture("flower2.jpg"); |
|  | //Picture smallP = pix.scale(0.25, 0.25); |
|  | //smallP.write("smallkai.jpg"); |
|  | pix.explore(); |
|  |  |
|  | //Picture p = new SimplePicture(); |
|  | } |
|  |  |
|  | } |

# Activity 4 Exercise Results

1. What was the output result after running the getCount method? 210  
   Paste your getCount method below.

|  |
| --- |
| public int getCount(int lookingfor){ |
|  |  | int timesfound = 0; |
|  |  | for(int row = 0; row < matrix.length; row++){ |
|  |  | for(int col = 0; col < matrix[row].length; col++){ |
|  |  | if(matrix[row][col]==(lookingfor)){ |
|  |  | timesfound++; |
|  |  | } |
|  |  | } |
|  |  | } |
|  |  | return timesfound; |
|  |  | } |

1. What is the output result after running the getLargest method? 310  
   Paste your getLargest method below.

|  |
| --- |
| public int getLargest(){ |
|  |  | int largest = -1; |
|  |  | for(int[] row : matrix){ |
|  |  | for(int i : row){ |
|  |  | if(i>largest){ |
|  |  | largest = i; |
|  |  | } |
|  |  | } |
|  |  | } |
|  |  | return largest; |
|  |  | } |

1. What is the output result after running the getColTotal method? 255  
   Paste your getColTotal method below.

|  |
| --- |
|  |
| public int getColTotal(int colLook){ |
|  |  | int total = 0; |
|  |  | for(int row = 0; row < matrix.length; row++){ |
|  |  | for(int col = 0; col < matrix[row].length; col++){ |
|  |  | if(col == colLook){ |
|  |  | total+=matrix[row][col]; |
|  |  | } |
|  |  | } |
|  |  | } |
|  |  | return total; |
|  |  | } |
|  |  |  |