Appendix D

All code produced by myself for the solution:

GUILayout1 Class

```java
package GUI;

import java.awt.BorderLayout;
import java.awt.Event;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.awt.image.RasterFormatException;
import java.io.File;
import java.io.IOException;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.KeyStroke;
import javax.swing.ScrollPaneConstants;
import tess4J.OCR;

public class GUILayout1 extends OCR {
        private JFrame frame, smallFrame; // private variables
        private JPanel contentPane, panel;
        private JButton crop, crop2, escape, grayscale, process, next, previous, select;
        private JLabel imgLabel;
        private ButtonGroup processButtonGroup;
        private static int FRAME_WIDTH = 615;
        private static int FRAME_HEIGHT = 150;
        private boolean isClicked = true;
        private JFileChooser fc = new JFileChooser();
        private File draftDraw;
        private int nextOrPrev = 0;
        private JScrollPane pane;

        public BufferedImage image, backupImage; // public variables and objects
        public String filePath, fileName, tessPath, tessName;
        public LinkedListQueue queue = new LinkedListQueue();
        public exportTags excel = new exportTags();
        public PDFconverter convert = new PDFconverter();

        public static void main(String[] args) { // main method
                GUILayout1 GUITabs = new GUILayout1();
                GUITabs.start();
        }

        private void start() { // Creates main frame and sets it to a default size
                frame = new JFrame("OCR Tag Checker");
```

```
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.pack();
		menuSetUp();
		makeContent();
		frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
		frame.setVisible(true);
		frame.setLocationRelativeTo(null);
		showHelp();
		JOptionPane.showMessageDialog(frame,
				"To see instructions again, refer to the 'USER GUIDE' menu option under 
the 'HELP' menu (top LEFT)",
				"HELP MENU", JOptionPane.INFORMATION_MESSAGE);
	}

	private void menuSetUp() { // Adds menu bars to main frame
		JMenuBar menuMain;
		menuMain = new JMenuBar();
		frame.setJMenuBar(menuMain);
		menuMain.add(dropdownMenu1());
		menuMain.add(dropdownMenu2());

	}

	private void makeContent() { // Adds first content pane and sets path for image
		contentPane = (JPanel) frame.getContentPane();
		contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));
		contentPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
		ImageIcon icon = new ImageIcon("path");
		JLabel label = new JLabel(icon);
		frame.add(label);
		SouthRegion();

	}

	private JMenu dropdownMenu1() { // creates items on the first menu bar
		JMenu menu;
		JMenuItem item;

		menu = new JMenu("File");
		menu.setMnemonic(KeyEvent.VK_F);

		item = new JMenuItem("Select");
		item.setMnemonic(KeyEvent.VK_S);
		item.addActionListener(new selectListener());
		item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, Event.ALT_MASK));
		menu.add(item);

		item = new JMenuItem("Export");
		item.setMnemonic(KeyEvent.VK_P);
		item.addActionListener(new exportListener());
		item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P, Event.ALT_MASK));
		menu.add(item);

		menu.addSeparator();

		item = new JMenuItem("Exit");
		item.setMnemonic(KeyEvent.VK_X);
		item.addActionListener(new exitListener());
		item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q, Event.CTRL_MASK));
		menu.add(item);

		return menu;
	}

	private JMenu dropdownMenu2() { // creates items on the second menu bar
		JMenu menu;
		JMenuItem item;

		menu = new JMenu("Help");
		menu.setMnemonic(KeyEvent.VK_H);

		item = new JMenuItem("User Guide");
		item.setMnemonic(KeyEvent.VK_G);
		item.addActionListener(new helpListener());
		item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_G, Event.ALT_MASK));
		menu.add(item);
		menu.addSeparator();

		item = new JMenuItem("Reset");
		item.setMnemonic(KeyEvent.VK_R);
		item.addActionListener(new resetListener());
		item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R, Event.ALT_MASK));
		menu.add(item);

		item = new JMenuItem("Close");
		item.setMnemonic(KeyEvent.VK_C);
		item.addActionListener(new clearListener());
		item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C, Event.ALT_MASK));
		menu.add(item);

		return menu;
	}

	private void SouthRegion() { // creates south region where all of the JButtons will be stored
		contentPane = (JPanel) frame.getContentPane();
		contentPane.setLayout(new BorderLayout(10, 10));
		panel = new JPanel();
		panel.setBorder(BorderFactory.createTitledBorder(
						"Please refer to the 'USER GUIDE' under the 'HELP' menu for additional
assistance and instructions!"));
		JPanel buttonPanel = new JPanel();

		processButtonGroup = new ButtonGroup();

		crop = new JButton("Crop");
		processButtonGroup.add(crop);
		crop.addActionListener(new cropButton());
		buttonPanel.add(crop);

		grayscale = new JButton("Grayscale");
		processButtonGroup.add(grayscale);
		grayscale.addActionListener(new grayscale());
		buttonPanel.add(grayscale);

		process = new JButton("Process");
		processButtonGroup.add(process);
		process.addActionListener(new process());
		processButtonGroup.add(process);
		buttonPanel.add(process);

		next = new JButton("Next");
		processButtonGroup.add(next);
		next.addActionListener(new next());
		processButtonGroup.add(process);
		buttonPanel.add(next);

		previous = new JButton("Previous");
		processButtonGroup.add(previous);
		previous.addActionListener(new previous());
		processButtonGroup.add(process);
		buttonPanel.add(previous);

		select = new JButton("Select");
		processButtonGroup.add(select);
		select.addActionListener(new select());
		processButtonGroup.add(process);
		buttonPanel.add(select);
```

```
                contentPane.add(panel, BorderLayout.CENTER);
                contentPane.add(buttonPanel, BorderLayout.SOUTH);

        }

        private void NorthRegion(String findFile) throws IOException { // adds file that is supposed to be
processed to the

                                // north

                                // region
                if (findFile.contains("\\")) {
                        findFile.replace("\\", "\\\\"); // ensures file has the double \\ required in
Java syntax
                }

                try {
                        if (findFile.contains(".pdf") || findFile.contains(".PDF")) { // checks if image
is a PDF
                                contentPane.remove(panel);
                                frame.setSize(1000, 1000); // frame size adjusts to appropriate size
                                frame.setLocationRelativeTo(null);
                                convert.generateImageFromPDF(findFile);
                                image = convert.generateImage(); // image is received from other class
                                imgLabel = new JLabel((new ImageIcon(image)), JLabel.CENTER);
                                contentPane.add(imgLabel, BorderLayout.NORTH);
                                crop.setEnabled(false);
                                grayscale.setEnabled(false);
                                process.setEnabled(false);
                                isClicked = false;
                                panel = new JPanel();
                                panel.setBorder(
                                                BorderFactory.createTitledBorder("Page " + (nextOrPrev +
1) + " out of " + convert.listSize())); // shows


                                        // PDF


                                        // page


                                        // user


                                        // is


                                        // on
                                contentPane.add(panel, BorderLayout.NORTH);
                        } else {
                                JOptionPane.showMessageDialog(frame, "Please ensure you have selected an
approved file type",
                                                "INCORRECT / NO FILE SELECTED",
JOptionPane.ERROR_MESSAGE);
                        }
                } catch (NullPointerException incorrectFile) {
                        JOptionPane.showMessageDialog(frame, "Please ensure you have selected a file",
"INCORRECT FILE",
                                        JOptionPane.ERROR_MESSAGE);
                        contentPane.removeAll();
                        SouthRegion();
                        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
                } catch (IllegalArgumentException incorrectFile2) {
                        contentPane.removeAll();
                        JOptionPane.showMessageDialog(frame, "Please ensure you have selected an approved
file type",
                                        "INCORRECT FILE", JOptionPane.ERROR_MESSAGE);
                        SouthRegion();
                        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
                } catch (IOException fileOpenProblem) {
                        contentPane.removeAll();
                        JOptionPane.showMessageDialog(frame,
                                        "Error opening/reading file. Please ensure the file is not
currently OPENED", "OPEN/READ ERROR",
                                        JOptionPane.ERROR_MESSAGE);
                        SouthRegion();
                        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
                }

        }

        private void showHelp() { // creates help menu for user (displayed at the start of the program)
                if (JOptionPane.showConfirmDialog(frame,
                                "Welcome to the OCR Tag Reader! To move onto instructions, press 'OK'.
\nIf you are ready to begin working, press 'CANCEL'.",
                                "WELCOME", JOptionPane.OK_CANCEL_OPTION) == JOptionPane.OK_OPTION) {
                        if (JOptionPane.showConfirmDialog(frame,
                                        "1. To select a file for processing, use the 'SELECT' option
under the 'FILE' \nmenu option (top LEFT)."
                                                        + "\n2. Select any PDF file! \n3. Once the file
is loaded in, use the 'NEXT' and 'PREVIOUS'\n buttons (BOTTOM) to cycle to the desired page."
                                                        + " \n4. Use the 'SELECT' button to choose the
page.",
                                        "SELECTING FILES", JOptionPane.OK_CANCEL_OPTION) ==
JOptionPane.OK_OPTION) {
                                if (JOptionPane.showConfirmDialog(frame,
                                                "1. Use the 'GRAYSCALE' button (BOTTOM) to remove all
color from the file and make it easier"
                                                                + "\nto process (this should be done if
the tags font/background is NOT black and white). "
                                                                + "\n2. Use the 'CROP' button (BOTTOM)
to crop the tag by dragging your cursor across it \nand then pressing the 'CROP' button once more.",
                                                "PRE-PROCESSING", JOptionPane.OK_CANCEL_OPTION) ==
JOptionPane.OK_OPTION) {
                                        if (JOptionPane.showConfirmDialog(frame,
                                                        "1. Once the tag has been captured, you can press
'PROCESS' (BOTTOM) to read \nand store the contents.\nAt this point, you may: "
                                                                        + "\n - Export the tag to an
Excel Spreadsheet using the 'EXPORT' menu option\n under 'FILE' option (top LEFT) "
                                                                        + "\n - Use the 'RESET' menu
option under the 'HELP' menu (top LEFT) and continue\n accumulating tags from the SAME DOCUMENT before
exporting "
                                                                        + "\n - Use the 'CLEAR' menu
option under the 'HELP' menu (top LEFT),\nselect a NEW DOCUMENT, and continue accumulating tags before
exporting",
                                                        "PROCESSING and EXPORTING",
JOptionPane.OK_CANCEL_OPTION) == JOptionPane.OK_OPTION) {
                                                if (JOptionPane.showConfirmDialog(frame,
                                                                "1. Located under the 'HELP' menu (top
LEFT), 'RESET' will clear all modifications made to a\ndisplayed file, "
                                                                                + "'CLEAR' will remove a
specified file and allow for the user to select a new one",
                                                                "RESET and CLEAR",
JOptionPane.OK_CANCEL_OPTION) == JOptionPane.OK_OPTION) {
                                                        if (JOptionPane.showConfirmDialog(frame,
                                                                        "To report any errors/new ideas
for future use, please contact the developer at \n********@gmail.com "
                                                                                        + "\nFuture
development ideas:",
                                                                        "ERRORS and FUTURE DEVELOPMENT",
                                                                        JOptionPane.OK_CANCEL_OPTION) ==
JOptionPane.OK_OPTION) {
                                                        }
                                                }
                                        }

                                }
```

```
				}
			}
		}

		private class resetListener implements ActionListener {
			public void actionPerformed(ActionEvent e) {
				if (image == null) {
					JOptionPane.showMessageDialog(frame, "No file selected, no need to
reset!", "RESET NOT NEEDED",
							JOptionPane.INFORMATION_MESSAGE);
				} else {
					contentPane.removeAll();
					image = null;
					imgLabel = null;
					pane = null;
					SouthRegion();
					grayscale.setEnabled(false);
					crop.setEnabled(false);
					process.setEnabled(false);
					panel = new JPanel();
					frame.setLocationRelativeTo(null);
					panel.setBorder(
							BorderFactory.createTitledBorder("Page " + (nextOrPrev +
1) + " out of " + convert.listSize()));
					contentPane.add(panel, BorderLayout.NORTH);
					convert.removeModImage(nextOrPrev, backupImage);
					image = convert.generateImage();
					imgLabel = new JLabel(new ImageIcon(image));
					pane = new JScrollPane(imgLabel);

		pane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);

		pane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
					contentPane.add(pane);
					frame.setSize(1000, 999);
					frame.setSize(1000, 1000);
					JOptionPane.showMessageDialog(frame, "Image reset!", "RESET SUCCESSFUL",
							JOptionPane.INFORMATION_MESSAGE);
					isClicked = false;
				}
			}
		}

		private class clearListener implements ActionListener { // clear all fields, including the
currently stored file
			public void actionPerformed(ActionEvent e) {
				if (image == null) {
					JOptionPane.showMessageDialog(frame, "No file selected, no need to
clear!", "CLEAR NOT NEEDED",
							JOptionPane.INFORMATION_MESSAGE);
				} else {
					frame.setLocationRelativeTo(null);
					contentPane.removeAll();
					convert.clearList();
					image = null;
					imgLabel = null;
					pane = null;
					fileName = null;
					nextOrPrev = 0;
					SouthRegion();
					frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
					JOptionPane.showMessageDialog(frame, "All fields cleared!", "CLEAR
SUCCESSFUL",
							JOptionPane.INFORMATION_MESSAGE);
					isClicked = false;
				}
			}
		}
```

```
	private class helpListener implements ActionListener { // Listener that will open the help guide
		public void actionPerformed(ActionEvent e) {
			showHelp();
		}
	}

	private class exportListener implements ActionListener { // outputs the data to a desired location
		public void actionPerformed(ActionEvent e) {
			if (queue.queueSize() == 0) {
				JOptionPane.showMessageDialog(frame, "Nothing to export!", "NOTHING TO 
EXPORT",
						JOptionPane.INFORMATION_MESSAGE);
			} else {
				try {
					fc.showSaveDialog(frame);
					File outputLocation = fc.getSelectedFile();
					String file_Location = outputLocation.getAbsolutePath();
					if (file_Location.contains("\\")) {
						file_Location.replace("\\", "\\\\");
					}
					if (!file_Location.contains(".xlsx")) {
						JOptionPane.showMessageDialog(frame, "Please ensure you 
are outputting to an Excel file",
								"INVALID OUTPUT LOCATION", 
JOptionPane.ERROR_MESSAGE);
					}

					else {
						excel.toExcel(queue, fileName, file_Location);
						JOptionPane.showMessageDialog(frame, "Tags uploaded 
successfully!", "UPLOAD SUCCESSFUL",
								JOptionPane.INFORMATION_MESSAGE);

					}
				} catch (NullPointerException nothing) {
					JOptionPane.showMessageDialog(frame, "Please ensure you have an 
image selected", "NO SELECTED FILE",
							JOptionPane.ERROR_MESSAGE);
				} catch (IOException e1) {
					JOptionPane.showMessageDialog(frame, "Error opening/reading 
file", "OPEN/READ ERROR",
							JOptionPane.ERROR_MESSAGE);
				}
			}
		}
	}

	private class exitListener implements ActionListener { // listener for the close option
		public void actionPerformed(ActionEvent e) {
			if (!queue.isEmpty()) {
				if (JOptionPane.showConfirmDialog(frame, "You have unsaved tags! Are you 
sure you want to exit?",
						"UNSAVED TAG(S)", JOptionPane.YES_NO_OPTION) == 
JOptionPane.YES_OPTION) {
					System.exit(0);

				}
			} else {
				System.exit(0);
			}
		}
	}

	private class selectListener implements ActionListener { // listener that selects the file that 
the user opens
		public void actionPerformed(ActionEvent e) {

			if (image != null) {
```

```
                                JOptionPane.showMessageDialog(frame,
                                                "File already selected! Please 'RESET' or 'CLEAR' before
continuing", "FILE ALREADY SELECTED",
                                                JOptionPane.INFORMATION_MESSAGE);
                        } else {
                                fc.showOpenDialog(frame);
                                draftDraw = fc.getSelectedFile();
                                if (draftDraw == null) {
                                        return;
                                }
                                filePath = draftDraw.getAbsolutePath();
                                fileName = draftDraw.getName();
                                try {
                                        NorthRegion(filePath);
                                        pane = new JScrollPane(imgLabel);

        pane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);

        pane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
                                        contentPane.add(pane);
                                } catch (IOException e1) {
                                        e1.printStackTrace();
                                } catch (NullPointerException wrongFile) {
                                        JOptionPane.showMessageDialog(frame, "Please ensure you have
selected an approved file type",
                                                        "INCORRECT FILE", JOptionPane.ERROR_MESSAGE);
                                }
                        }
                }
        }

        private class cropButton implements ActionListener { // listener for the crop method
                public void actionPerformed(ActionEvent e) {
                        if (image == null) {
                                JOptionPane.showMessageDialog(frame, "Please ensure you have an image
selected", "NO SELECTED FILE",
                                                JOptionPane.ERROR_MESSAGE);
                        } else {
                                cropClassMain cropMethod = new cropClassMain(image);
                                smallFrame = new JFrame("Crop"); // seperate frame is created for
cropping of the image
                                smallFrame.pack();
                                smallFrame.setSize(1000, 1000);
                                smallFrame.setLayout(new BorderLayout());
                                smallFrame.setLocationRelativeTo(null);

                                JScrollPane scrolly = new JScrollPane(cropMethod,
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
                                                ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
                                smallFrame.add(scrolly);

                                JPanel buttonPanel = new JPanel();
                                crop2 = new JButton("Crop");
                                escape = new JButton("Escape");
                                buttonPanel.add(crop2);
                                buttonPanel.add(escape);
                                smallFrame.add(buttonPanel, BorderLayout.SOUTH);
                                smallFrame.setVisible(true);
                                contentPane.setSize(990, 990);

                                crop2.addActionListener(new ActionListener() {
                                        public void actionPerformed(ActionEvent e) {
                                                try {
                                                        contentPane.removeAll();
                                                        image = cropMethod.croppedImage();
                                                        imgLabel = new JLabel(new ImageIcon(image));
                                                        contentPane.add(imgLabel);
                                                        frame.setSize(1000, 999); // frame size adjusts
to image size
```

```
                                                        frame.setSize(1000, 1000);
                                                        pane = new JScrollPane(imgLabel);

        pane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);

        pane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
                                                        SouthRegion();
                                                        next.setEnabled(false);
                                                        select.setEnabled(false);
                                                        previous.setEnabled(false);
                                                        contentPane.add(pane);
                                                        smallFrame.dispose();
                                                        isClicked = true;
                                                        panel = new JPanel();
                                                        panel.setBorder(BorderFactory
                                                                        .createTitledBorder("Page " +
(nextOrPrev + 1) + " out of " + convert.listSize()));
                                                        contentPane.add(panel, BorderLayout.NORTH);
                                                } catch (RasterFormatException rast) {
                                                        JOptionPane.showMessageDialog(smallFrame, "Please
ensure your crop is within range!",
                                                                        "CROP OUT OF RANGE",
JOptionPane.ERROR_MESSAGE);
                                                }
                                        }
                                });

                                escape.addActionListener(new ActionListener() {
                                        public void actionPerformed(ActionEvent e) {
                                                smallFrame.dispose();
                                        }
                                });
                        }
                }
        }

        private class grayscale implements ActionListener { // converts image to grayscale
                public void actionPerformed(ActionEvent e) {
                        if (image == null) {
                                JOptionPane.showMessageDialog(frame, "Please ensure you have an image
selected", "NO SELECTED FILE",
                                                JOptionPane.ERROR_MESSAGE);
                        } else {
                                try {
                                        grayscaleFunc operate = new grayscaleFunc(image);
                                        contentPane.removeAll();
                                        image = operate.imageGray();
                                        imgLabel = new JLabel(new ImageIcon(image));
                                        contentPane.add(imgLabel);
                                        frame.setSize(1000, 999); // frame size adjusts to image size
                                        frame.setSize(1000, 1000);
                                        pane = new JScrollPane(imgLabel);

        pane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);

        pane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
                                        SouthRegion();
                                        next.setEnabled(false);
                                        select.setEnabled(false);
                                        previous.setEnabled(false);
                                        panel = new JPanel();
                                        panel.setBorder(BorderFactory
                                                        .createTitledBorder("Page " + (nextOrPrev + 1) +
" out of " + convert.listSize()));
                                        contentPane.add(panel, BorderLayout.NORTH);
                                        contentPane.add(pane);
                                } catch (NullPointerException nothing) {
                                        JOptionPane.showMessageDialog(frame, "Please ensure you have an
image selected", "NO SELECTED FILE",
								JOptionPane.ERROR_MESSAGE);
				}
			}
		}
	}

	private class next implements ActionListener { // if multiple pages, cycle to the right one
		public void actionPerformed(ActionEvent e) {
			if (image == null) {
				JOptionPane.showMessageDialog(frame, "Please ensure you have an image 
selected", "NO SELECTED FILE",
						JOptionPane.ERROR_MESSAGE);
			} else {
				if (convert.listSize() == 1) {
					JOptionPane.showMessageDialog(frame, "This file contains only 1 
page!", "NO MORE PAGES",
							JOptionPane.INFORMATION_MESSAGE);
				} else {
					contentPane.removeAll();
					panel = new JPanel();
					panel.setBorder(BorderFactory
							.createTitledBorder("Page " + (nextOrPrev + 1) + 
" out of " + convert.listSize()));
					contentPane.add(panel, BorderLayout.NORTH);
					nextOrPrev++;
					if (nextOrPrev >= convert.listSize()) {
						nextOrPrev = 0;
					}
					image = convert.nextOrPrev(nextOrPrev);
					imgLabel = new JLabel(new 
ImageIcon(convert.nextOrPrev(nextOrPrev)));
					contentPane.add(imgLabel);
					panel.setBorder(BorderFactory
							.createTitledBorder("Page " + (nextOrPrev + 1) + 
" out of " + convert.listSize()));
					frame.setSize(1000, 999); // frame size adjusts to image size
					frame.setSize(1000, 1000);
					pane = new JScrollPane(imgLabel);

	pane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);

	pane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
					SouthRegion();
					grayscale.setEnabled(false);
					crop.setEnabled(false);
					process.setEnabled(false);
					panel = new JPanel();
					panel.setBorder(BorderFactory
							.createTitledBorder("Page " + (nextOrPrev + 1) + 
" out of " + convert.listSize()));
					contentPane.add(panel, BorderLayout.NORTH);
					contentPane.add(pane);
				}
			}
		}
	}

	private class previous implements ActionListener { // if multiple pages, cycle to the right one
		public void actionPerformed(ActionEvent e) {
			if (image == null) {
				JOptionPane.showMessageDialog(frame, "Please ensure you have an image 
selected", "NO SELECTED FILE",
						JOptionPane.ERROR_MESSAGE);
			} else {
				if (convert.listSize() == 1) {
					JOptionPane.showMessageDialog(frame, "This file contains only 1 
page!", "NO MORE PAGES",
							JOptionPane.INFORMATION_MESSAGE);
				} else {
```

```java
                    contentPane.removeAll();
                    nextOrPrev--;
                    if (nextOrPrev == -1) {
                        nextOrPrev = convert.listSize() - 1;
                    }
                    image = convert.nextOrPrev(nextOrPrev);
                    imgLabel = new JLabel(new
ImageIcon(convert.nextOrPrev(nextOrPrev)));
                    contentPane.add(imgLabel);
                    frame.setSize(1000, 999); // frame size adjusts to image size
                    frame.setSize(1000, 1000);
                    pane = new JScrollPane(imgLabel);

    pane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);

    pane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
                    SouthRegion();
                    grayscale.setEnabled(false);
                    crop.setEnabled(false);
                    process.setEnabled(false);
                    panel = new JPanel();
                    panel.setBorder(BorderFactory
                            .createTitledBorder("Page " + (nextOrPrev + 1) +
" out of " + convert.listSize()));
                    contentPane.add(panel, BorderLayout.NORTH);
                    contentPane.add(pane);
                }
            }
        }

    }

    private class select implements ActionListener { // if multiple pages, cycle to the right one
        public void actionPerformed(ActionEvent e) {
            if (image == null) {
                JOptionPane.showMessageDialog(frame, "Please ensure you have an image
selected", "NO SELECTED FILE",
                        JOptionPane.ERROR_MESSAGE);
            } else {
                image = convert.nextOrPrev(nextOrPrev);
                backupImage = convert.nextOrPrev(nextOrPrev);
                frame.setSize(1000, 999);
                frame.setSize(1000, 1000);
                grayscale.setEnabled(true);
                crop.setEnabled(true);
                process.setEnabled(true);
                next.setEnabled(false);
                previous.setEnabled(false);
                select.setEnabled(false);
                JOptionPane.showMessageDialog(frame, "Page selected successfully!", "PAGE
SELECTION",
                        JOptionPane.INFORMATION_MESSAGE);
            }
        }
    }

    private class process implements ActionListener { // outputs the data to a desired location
        public void actionPerformed(ActionEvent e) {
            if (image == null) {
                JOptionPane.showMessageDialog(frame, "Please ensure you have an image
selected", "NO SELECTED FILE",
                        JOptionPane.ERROR_MESSAGE);
            } else {
                if (isClicked == true) {
                    try {
                        process.setEnabled(false);
                        crop.setEnabled(false);
                        grayscale.setEnabled(false);
                        queue.enqueue(fileName);
```

```
                                                queue.enqueue(fileName(image));
                                                JOptionPane.showMessageDialog(frame,
                                                                "Tag added to queue! Please press
'EXPORT' when you are ready to move all tags to Excel!",
                                                                "TAG ADDED",
JOptionPane.INFORMATION_MESSAGE);

                                        } catch (NullPointerException nothing) {
                                                JOptionPane.showMessageDialog(frame, "Please ensure you
have an image selected",
                                                                "NO SELECTED FILE",
JOptionPane.ERROR_MESSAGE);
                                        }
                                } else {
                                        if ((JOptionPane.showConfirmDialog(frame,
                                                        "Are you sure you want to process the entire
page? This is not recommended.",
                                                        "FILE NOT CROPPED", JOptionPane.YES_NO_OPTION))
== JOptionPane.YES_OPTION) {
                                                queue.enqueue(fileName);
                                                queue.enqueue(fileName(image));
                                                process.setEnabled(false);
                                                crop.setEnabled(false);
                                                grayscale.setEnabled(false);
                                                JOptionPane.showMessageDialog(frame,
                                                                "Tag added to queue! Please press
'EXPORT' when you are ready to move all tags to Excel!",
                                                                "TAG ADDED",
JOptionPane.INFORMATION_MESSAGE);
                                        }
                                }
                        }
                }
        }
}
```

cropClassMain Class

```
package GUI;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;

public class cropClassMain extends JPanel {
        private int x, y, w, h;
        private BufferedImage image, image2;
        private Rectangle cropRec;
        private Rectangle finalRect;

        protected cropClassMain(BufferedImage image) {
                this.image = image;
                MouseHandler mouse = new MouseHandler();
                addMouseListener(mouse);
                addMouseMotionListener(mouse);
        }

        public Dimension getPreferredSize() { //formats the size of the cropping window (for the
JScrollPane)
                return new Dimension(image.getWidth(), image.getHeight());
        }
        private Rectangle getCropBounds() { //returns rectangle with boundaries from the mouse movements
                finalRect = null;
                if (cropRec != null) {
                        x = cropRec.x;
                        y = cropRec.y;
                        w = cropRec.width;
                        h = cropRec.height;
                }
                finalRect = new Rectangle(x, y, w, h);
                return finalRect;
        }

        protected void paintComponent(Graphics g) { //sets image as background and allows for drawing of a
cropping rect.
                Rectangle drawCrop = getCropBounds();
                super.paintComponent(g);
                if (drawCrop != null) {
                        g.drawImage(image, 0, 0, null);
                        g.setColor(Color.red);
                        g.drawRect(x, y, w, h);
                }
        }

        protected class MouseHandler extends MouseAdapter { //allows for mouse input and sets the
coordinates
                public void mouseReleased(MouseEvent a) {
                        cropRec = null;
                        repaint();
                }

                public void mousePressed(MouseEvent a) {
                        cropRec = new Rectangle();
                        cropRec.setLocation(a.getPoint());
                        repaint();
                }

                public void mouseDragged(MouseEvent a) {
                        Point point = a.getPoint();
                        int recWidth = point.x - cropRec.x;
                        int recHeight = point.y - cropRec.y;
                        cropRec.setSize(recWidth, recHeight);
                        repaint();
                }
        }

        protected BufferedImage croppedImage() { //uses rectangle coordinates/length/width to return
subimage
                image2 = image.getSubimage(Math.abs(x), Math.abs(y), Math.abs(w), Math.abs(h));
                return image2;
        }

}
```

grayscaleFunc Class

```
package GUI;

import java.awt.Color;
import java.awt.image.BufferedImage;

public class grayscaleFunc {
        protected grayscaleFunc(BufferedImage image) {
                this.image = image;
        }

        private int getHeight() { //important for the conversion
                return image.getHeight();
```

```
	}

	private int getWidth() { //important for the conversion
		return image.getWidth();
	}

	protected BufferedImage imageGray() { // changes each image pixel to grey using nested for loop
and an RGB
	                                                                        // calculation
		if (image == null) {
			return null;
		}
		for (int i = 0; i < getHeight(); i++) {
			for (int j = 0; j < getWidth(); j++) {
				Color imageColor = new Color(image.getRGB(j, i));
				int rgb = ((int) (imageColor.getRed()) + (int) (imageColor.getGreen()) +
(int) (imageColor.getBlue()))
				                 / 3;
				Color newColor = new Color(rgb, rgb, rgb);
				image.setRGB(j, i, newColor.getRGB());
			}
		}
		return image;
	}

	private BufferedImage image;
}
```

PDFconverter Class

```
package GUI;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.varia.NullAppender;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.encryption.InvalidPasswordException;
import org.apache.pdfbox.rendering.ImageType;
import org.apache.pdfbox.rendering.PDFRenderer;

public class PDFconverter {
	private BufferedImage img;
	private int pageCount;
	public ArrayList<BufferedImage> storageList = new ArrayList<BufferedImage>();

	public void generateImageFromPDF(String convert) throws InvalidPasswordException, IOException {
//converts all pages to images and stores them in an ArrayList
		PDDocument doc = PDDocument.load(new File(convert));
		PDFRenderer pdfRender = new PDFRenderer(doc);
		pageCount = doc.getNumberOfPages();
		for (int page = 0; page < pageCount; ++page) {
			img = pdfRender.renderImageWithDPI(page, 300, ImageType.RGB);
			storageList.add(img);
		}
		doc.close();
	}

	protected int listSize() { //size of list
		return storageList.size();
	}

	protected void clearList() { //empties the list when called on
```

```
		storageList.clear();
	}

	protected BufferedImage generateImage() { //returns first page
		return storageList.get(0);
	}

	protected void removeModImage(int nextOrPrev, BufferedImage backupImage) { //used for resetting
		storageList.remove(nextOrPrev);
		storageList.add(nextOrPrev, backupImage);
	}

	protected BufferedImage nextOrPrev(int nextOrPrev) { //cycles between the pages
		return storageList.get(nextOrPrev);
	}
}
```

QueueInterface Interface

```
package GUI;

public interface QueueInterface {

	boolean isEmpty(); //determines if queue is empty

	void enqueue (Object obj); //adds object to the end of the queue

	Object dequeue(); //removes and returns first object in the queue

	Object peekFront(); //returns first object in queue without removing it

	Object peekEnd(); //returns last object in queue without removing it

	int queueSize(); //returns size of queue

}
```

LinkedListQueue Class

```
package GUI;

import java.util.LinkedList;

public class LinkedListQueue implements QueueInterface {

	public LinkedListQueue() {
		lst = new LinkedList();
	}

	public boolean isEmpty() {
		return lst.isEmpty();
	}

	public void enqueue(Object obj) {
		lst.addLast(obj);
	}

	public Object dequeue() {
		if (queueSize() > 0) {
			return lst.removeFirst();
		} else {
		throw new NullPointerException();
		}
	}

	public Object peekFront() {
```

```
                if (queueSize() > 0) {
                        return lst.getFirst();
                } else {
                throw new NullPointerException();
                }
        }

        public Object peekEnd() {
                if (queueSize() > 0) {
                        return lst.getLast();
                } else {
                throw new NullPointerException();
                }
        }

        public int queueSize() {
                return lst.size();
        }

        private LinkedList lst;

}
```

exportTags class

```
package GUI;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Iterator;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;


public class exportTags {
        public void toExcel(LinkedListQueue queue, String tagFile, String fileLocation) throws IOException
{
                try {

                        int dataSize = 0;
                        LinkedListQueue storageQueue = new LinkedListQueue(); // new queue

                        FileInputStream checkValues = new FileInputStream(fileLocation); // object to
read and input file contents

                        XSSFWorkbook workbook = new XSSFWorkbook(); // new workbook for output
                        XSSFSheet sheet = workbook.createSheet("output"); // new sheet for output

                        XSSFWorkbook input = new XSSFWorkbook(fileLocation); // new workbook for input
                        XSSFSheet inputSheet = input.getSheetAt(0); // new sheet for input

                        LocalDateTime time = LocalDateTime.now(); // time for output
                        DateTimeFormatter format = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

                        Object[][] tagData = new Object[1000][3]; // 2D array for output

                        int rowNum, columnNum;
                        Iterator<Row> rowCheck = inputSheet.iterator();
                        while (rowCheck.hasNext()) { // iterate through each row
                                Row row = rowCheck.next();
                                Iterator<Cell> cellIterator = row.cellIterator(); // iterate through each
cell
                                while (cellIterator.hasNext()) {
```

```
                    Cell cell = cellIterator.next();
                    switch (cell.getCellType()) {
                    case NUMERIC:
                        storageQueue.enqueue(cell.getNumericCellValue());
                        break;
                    case STRING:
                        storageQueue.enqueue(cell.getStringCellValue());
                        break;
                    case BLANK:
                        break;
                    default:
                        break;
                    }

                }
            }
            checkValues.close();
            input.close();

            if (storageQueue.queueSize() > 0) {
                for (int x = 0; x <= storageQueue.queueSize(); x++) {
                    tagData[x][0] = storageQueue.dequeue();
                    tagData[x][1] = storageQueue.dequeue();
                    tagData[x][2] = storageQueue.dequeue();
                    dataSize = x;
                }
            }

            for (int x = dataSize+1; x <= queue.queueSize() + dataSize; x++) { // all tags
dequeued into the 2D

                                // array
                tagData[x][0] = queue.dequeue();
                tagData[x][1] = queue.dequeue();
                tagData[x][2] = time.format(format);
            }

            rowNum = 0;

            for (Object[] tag : tagData) { // nested for-each loop
                Row row = sheet.createRow(++rowNum); // creates required rows for sheet
                columnNum = 0;

                for (Object field : tag) {
                    Cell cell = row.createCell(++columnNum); // creates required
columns for sheets
                    if (field instanceof String) { // if String, fill sheet with
String

                        cell.setCellValue((String) field);
                    } else if (field instanceof Integer) { // if int, fill cell with
int

                        cell.setCellValue((Integer) field);
                    }
                }

            }
            sheet.autoSizeColumn(0);
            sheet.autoSizeColumn(1);
            sheet.autoSizeColumn(2);
            sheet.autoSizeColumn(3);

            try (FileOutputStream outputStream = new FileOutputStream(fileLocation)) {
                workbook.write(outputStream); // output to selected file
            }
            workbook.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
        }
}
```

OCR class

```
package tess4J;

import java.awt.image.BufferedImage;
import net.sourceforge.tess4j.Tesseract;
import net.sourceforge.tess4j.TesseractException;

public class OCR { // OCR Scanner
        private String text;
        Tesseract tesseract = new Tesseract();

        protected String fileName(BufferedImage large) {
                {
                        try {
                                tesseract.setDatapath("C:\\Users\\rianr\\Desktop\\Tess4J\\tessdata"); //
sets path

                                text = tesseract.doOCR(large); // scans the image

                        } catch (TesseractException e) { // catches custom errors related to the
Tesseract engine
                                e.printStackTrace();
                        }
                }
                return text;
        }
}
```