Learning Journal Unit\_6

Dawid Blom

**Thursday, 13/05/2021, 7 – 12 am: Reading Assignment:** This week in the reading assignment I found the Socket programming quite interesting since it’s an area that I was struggling with. However, after reading how Java does it, it started to make more sense. Therefore, I cannot wait for the holiday when I have more time so that I can start getting into some socket programming in C++.

**Thursday, 13/05/2021, 1 – 2 pm: Discussion Assignment:** The discussion topic for this week was an area I really enjoy in programming. Socket and server/client based programming is of high interest to me. Furthermore, I decided to make my discussion a little bit more playful and humorous than my previous contributions. I just found that I could explain the relationship of Server Sockets and Sockets in general a lot better when giving it a face other than a technology face.

**Thursday, 13/05/2021, 2:30 – 3 pm: Self-Quiz:** The self quiz of this week was good. It really helped me to establish the knowledge I gained from reading the book a lot better. Furthermore, I kept the book open while I did the self quiz because I really wanted to get full marks on the first go. I was able to reach my goal. By looking through the book as I did the self quiz I was able to dig a lot deeper into the book and I was able to gain even more knowledge from it.

**Thursday, 13/05/2021, 3 – 3:40 pm: Graded-Quiz:** So this week was graded quiz. I am completely astonished by the marks I got for it. I am so proud and happy at the moment. Furthermore, it truly tested my knowledge as well as the amount of information I was able to obtain and the critical thinking I had to apply as I went through the graded quiz.

**Monday, 17/05/2021, 8 – 12 pm: Programming Assignment:** The description of this week’s programming assignment was a lot clearer. Moreover, the save and open file features that was required to be implemented was really fun and it helped me better understand how files work in Java specifically. Furthermore, for my added feature I wanted to add a colour chooser, however, I quickly realised that it was already implemented in the program. Therefore, I decided to implement a font changer for my added feature to the program. I am quite happy how it turned out.

package textcollage;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.RenderingHints;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.image.BufferedImage;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.io.Reader;

import java.util.ArrayList;

import java.util.Scanner;

import javax.imageio.ImageIO;

import javax.swing.BorderFactory;

import javax.swing.BoxLayout;

import javax.swing.DefaultListCellRenderer;

import javax.swing.JColorChooser;

import javax.swing.JComboBox;

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.JTextField;

import javax.swing.KeyStroke;

/\*\*

\* A panel that contains a large drawing area where strings

\* can be drawn. The strings are represented by objects of

\* type DrawTextItem. An input box under the panel allows

\* the user to specify what string will be drawn when the

\* user clicks on the drawing area.

\*/

public class DrawTextPanel extends JPanel{

// As it now stands, this class can only show one string at at

// a time! The data for that string is in the DrawTextItem object

// named theString. (If it's null, nothing is shown. This

// variable should be replaced by a variable of type

// ArrayList<DrawStringItem> that can store multiple items.

private ArrayList<DrawTextItem> theString; // change to an ArrayList<DrawTextItem> !

// Combo boxes for string font options.

private JComboBox<String> fontChoices;

private JComboBox<String> fontStyles;

private JComboBox<String> fontSizes;

private JPanel fontOptions;

private Color currentTextColor = Color.BLACK; // Color applied to new strings.

private Canvas canvas; // the drawing area.

private JTextField input; // where the user inputs the string that will be added to the canvas

private SimpleFileChooser fileChooser; // for letting the user select files

private JMenuBar menuBar; // a menu bar with command that affect this panel

private MenuHandler menuHandler; // a listener that responds whenever the user selects a menu command

private JMenuItem undoMenuItem; // the "Remove Item" command from the edit menu

/\*\*

\* An object of type Canvas is used for the drawing area.

\* The canvas simply displays all the DrawTextItems that

\* are stored in the ArrayList, strings.

\*/

private class Canvas extends JPanel {

Canvas() {

setPreferredSize( new Dimension(800,600) );

setBackground(Color.LIGHT\_GRAY);

setFont( new Font( "Serif", Font.BOLD, 24 ));

}

protected void paintComponent(Graphics g) {

super.paintComponent(g);

((Graphics2D)g).setRenderingHint(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

if (theString != null)

for (DrawTextItem i : theString)

i.draw(g);

}

}

/\*\*

\* An object of type MenuHandler is registered as the ActionListener

\* for all the commands in the menu bar. The MenuHandler object

\* simply calls doMenuCommand() when the user selects a command

\* from the menu.

\*/

private class MenuHandler implements ActionListener {

public void actionPerformed(ActionEvent evt) {

doMenuCommand( evt.getActionCommand());

}

}

/\*\*

\* Creates a DrawTextPanel. The panel has a large drawing area and

\* a text input box where the user can specify a string. When the

\* user clicks the drawing area, the string is added to the drawing

\* area at the point where the user clicked.

\*/

public DrawTextPanel() {

fileChooser = new SimpleFileChooser();

undoMenuItem = new JMenuItem("Remove Item");

undoMenuItem.setEnabled(false);

menuHandler = new MenuHandler();

setLayout(new BorderLayout(3,3));

setBackground(Color.BLACK);

setBorder(BorderFactory.createLineBorder(Color.BLACK, 2));

canvas = new Canvas();

add(canvas, BorderLayout.CENTER);

JPanel bottom = new JPanel();

bottom.add(new JLabel("Text to add: "));

input = new JTextField("Hello World!", 40);

bottom.add(input);

add(bottom, BorderLayout.SOUTH);

fontOptions = new JPanel();

fontOptions.setLayout(new BoxLayout(fontOptions, BoxLayout.Y\_AXIS));

JPanel titleBar = new JPanel();

titleBar.setMaximumSize(new Dimension(200, 75));

JLabel fontTitle = new JLabel("Font Options");

fontTitle.setFont(new Font("Serif", Font.BOLD, 24));

titleBar.add(fontTitle);

fontOptions.add(titleBar);

// Used to center the text in a combo box

DefaultListCellRenderer dlcr = new DefaultListCellRenderer();

dlcr.setHorizontalAlignment(DefaultListCellRenderer.CENTER);

fontChoices = new JComboBox<String>();

fontChoices.addItem("Serif");

fontChoices.addItem("SansSerif");

fontChoices.addItem("Dialog");

fontChoices.addItem("Monospaced");

fontChoices.setPreferredSize(new Dimension(100, 25));

fontChoices.setRenderer(dlcr);

JPanel fontTypeSelection = new JPanel();

fontTypeSelection.add(new JLabel("Type"));

fontTypeSelection.add(fontChoices);

fontTypeSelection.setMaximumSize(new Dimension(150, 50));

fontOptions.add(fontTypeSelection);

add(fontOptions, BorderLayout.EAST);

fontStyles = new JComboBox<String>();

fontStyles.addItem("Plain");

fontStyles.addItem("Bold");

fontStyles.addItem("Italic");

fontStyles.addItem("Bold+Italic");

fontStyles.setPreferredSize(new Dimension(100, 25));

fontStyles.setRenderer(dlcr);

JPanel fontStyleSelection = new JPanel();

fontStyleSelection.add(new JLabel("Style"));

fontStyleSelection.add(fontStyles);

fontStyleSelection.setMaximumSize(new Dimension(150, 50));

fontOptions.add(fontStyleSelection);

add(fontOptions, BorderLayout.EAST);

fontSizes = new JComboBox<String>();

fontSizes.addItem("12");

fontSizes.addItem("16");

fontSizes.addItem("24");

fontSizes.addItem("36");

fontSizes.addItem("48");

fontSizes.addItem("72");

fontSizes.setPreferredSize(new Dimension(100, 25));

fontSizes.setRenderer(dlcr);

JPanel fontSizeSelection = new JPanel();

fontSizeSelection.add(new JLabel("Size"));

fontSizeSelection.add(fontSizes);

fontSizeSelection.setMaximumSize(new Dimension(150, 50));

fontOptions.add(fontSizeSelection);

add(fontOptions, BorderLayout.EAST);

canvas.addMouseListener( new MouseAdapter() {

public void mousePressed(MouseEvent e) {

doMousePress( e );

}

} );

}

/\*\*

\* This method is called when the user clicks the drawing area.

\* A new string is added to the drawing area. The center of

\* the string is at the point where the user clicked.

\* @param e the mouse event that was generated when the user clicked

\*/

public void doMousePress( MouseEvent e ) {

String text = input.getText().trim();

if (text.length() == 0) {

input.setText("Hello World!");

text = "Hello World!";

}

DrawTextItem s = new DrawTextItem( text, e.getX(), e.getY() );

s.setTextColor(currentTextColor); // Default is null, meaning default color of the canvas (black).

// SOME OTHER OPTIONS THAT CAN BE APPLIED TO TEXT ITEMS:

s.setFont( getCurrentFont()); // Default is null, meaning font of canvas.

// s.setMagnification(3); // Default is 1, meaning no magnification.

// s.setBorder(true); // Default is false, meaning don't draw a border.

// s.setRotationAngle(25); // Default is 0, meaning no rotation.

// s.setTextTransparency(0.3); // Default is 0, meaning text is not at all transparent.

s.setBackground(Color.CYAN); // Default is null, meaning don't draw a background area.

// s.setBackgroundTransparency(0.7); // Default is 0, meaning background is not transparent.

if (theString == null)

theString = new ArrayList<DrawTextItem>();

theString.add(s); // Set this string as the ONLY string to be drawn on the canvas!

undoMenuItem.setEnabled(true);

canvas.repaint();

}

/\*\*

\* Returns a menu bar containing commands that affect this panel. The menu

\* bar is meant to appear in the same window that contains this panel.

\*/

public JMenuBar getMenuBar() {

if (menuBar == null) {

menuBar = new JMenuBar();

String commandKey; // for making keyboard accelerators for menu commands

if (System.getProperty("mrj.version") == null)

commandKey = "control "; // command key for non-Mac OS

else

commandKey = "meta "; // command key for Mac OS

JMenu fileMenu = new JMenu("File");

menuBar.add(fileMenu);

JMenuItem saveItem = new JMenuItem("Save...");

saveItem.setAccelerator(KeyStroke.getKeyStroke(commandKey + "N"));

saveItem.addActionListener(menuHandler);

fileMenu.add(saveItem);

JMenuItem openItem = new JMenuItem("Open...");

openItem.setAccelerator(KeyStroke.getKeyStroke(commandKey + "O"));

openItem.addActionListener(menuHandler);

fileMenu.add(openItem);

fileMenu.addSeparator();

JMenuItem saveImageItem = new JMenuItem("Save Image...");

saveImageItem.addActionListener(menuHandler);

fileMenu.add(saveImageItem);

JMenu editMenu = new JMenu("Edit");

menuBar.add(editMenu);

undoMenuItem.addActionListener(menuHandler); // undoItem was created in the constructor

undoMenuItem.setAccelerator(KeyStroke.getKeyStroke(commandKey + "Z"));

editMenu.add(undoMenuItem);

editMenu.addSeparator();

JMenuItem clearItem = new JMenuItem("Clear");

clearItem.addActionListener(menuHandler);

editMenu.add(clearItem);

JMenu optionsMenu = new JMenu("Options");

menuBar.add(optionsMenu);

JMenuItem colorItem = new JMenuItem("Set Text Color...");

colorItem.setAccelerator(KeyStroke.getKeyStroke(commandKey + "T"));

colorItem.addActionListener(menuHandler);

optionsMenu.add(colorItem);

JMenuItem bgColorItem = new JMenuItem("Set Background Color...");

bgColorItem.addActionListener(menuHandler);

optionsMenu.add(bgColorItem);

}

return menuBar;

}

/\*\*

\* Carry out one of the commands from the menu bar.

\* @param command the text of the menu command.

\*/

private void doMenuCommand(String command) {

if (command.equals("Save...")) { // save all the string info to a file

saveFile();

}

else if (command.equals("Open...")) { // read a previously saved file, and reconstruct the list of strings

openFile();

canvas.repaint(); // (you'll need this to make the new list of strings take effect)

}

else if (command.equals("Clear")) { // remove all strings

theString = null; // Remove the ONLY string from the canvas.

undoMenuItem.setEnabled(false);

canvas.repaint();

}

else if (command.equals("Remove Item")) { // remove the most recently added string

theString = null; // Remove the ONLY string from the canvas.

undoMenuItem.setEnabled(false);

canvas.repaint();

}

else if (command.equals("Set Text Color...")) {

Color c = JColorChooser.showDialog(this, "Select Text Color", currentTextColor);

if (c != null)

currentTextColor = c;

}

else if (command.equals("Set Background Color...")) {

Color c = JColorChooser.showDialog(this, "Select Background Color", canvas.getBackground());

if (c != null) {

canvas.setBackground(c);

canvas.repaint();

}

}

else if (command.equals("Save Image...")) { // save a PNG image of the drawing area

File imageFile = fileChooser.getOutputFile(this, "Select Image File Name", "textimage.png");

if (imageFile == null)

return;

try {

// Because the image is not available, I will make a new BufferedImage and

// draw the same data to the BufferedImage as is shown in the panel.

// A BufferedImage is an image that is stored in memory, not on the screen.

// There is a convenient method for writing a BufferedImage to a file.

BufferedImage image = new BufferedImage(canvas.getWidth(),canvas.getHeight(),

BufferedImage.TYPE\_INT\_RGB);

Graphics g = image.getGraphics();

g.setFont(canvas.getFont());

canvas.paintComponent(g); // draws the canvas onto the BufferedImage, not the screen!

boolean ok = ImageIO.write(image, "PNG", imageFile); // write to the file

if (ok == false)

throw new Exception("PNG format not supported (this shouldn't happen!).");

}

catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Sorry, an error occurred while trying to save the image:\n" + e);

}

}

}

/\*\*

\* By calling this method, it should

\* save the image created in text form to a file.

\* If the user did not choose a file for the contents

\* to be saved to, then a message will state to the user

\* to choose a file.

\*/

private void saveFile()

{

File save = fileChooser.getOutputFile(this, "Save File", "TextImage.txt");

if (save == null)

return;

PrintWriter out;

try

{

FileWriter stream = new FileWriter(save);

out = new PrintWriter(stream);

Color background = canvas.getBackground();

for (DrawTextItem i : theString)

{

Color textColor = i.getTextColor();

Font textFont = i.getFont();

out.println(background.getRed() + " " + background.getGreen() + " " + background.getBlue() + "$" +

i.getString() + "$" +

textColor.getRed() + " " + textColor.getGreen() + " " + textColor.getBlue() + "$" +

i.getX() + "$" +

i.getY() + "$" +

textFont.getName() + "$" +

textFont.getStyle() + "$" +

textFont.getSize());

out.flush();

}

out.close();

}

catch (IOException e)

{

JOptionPane.showMessageDialog(this, "Error: please specify a file name to be used." + e);

System.out.println("Error: please specify a filename to be used." + e);

}

}

/\*\*

\* By calling this method, it should

\* open a previously saved file from this

\* program to reconstruct the image saved.

\* However, if the user did not specify a

\* file to open or if it's not in the right

\* format then a message will appear to state

\* such error.

\*/

private void openFile()

{

File open = fileChooser.getInputFile(this, "Open");

if (open == null)

return;

Scanner in;

try

{

Reader stream = new BufferedReader(new FileReader(open));

in = new Scanner(stream);

ArrayList<DrawTextItem> newString = new ArrayList<DrawTextItem>();

while (in.hasNextLine())

{

String item = in.nextLine();

String[] properties = item.split("\\$");

DrawTextItem tempItem = new DrawTextItem(properties[1]);

String[] bgColor = properties[0].split(" ");

tempItem.setBackground(new Color(Integer.parseInt(bgColor[0]),

Integer.parseInt(bgColor[1]), Integer.parseInt(bgColor[2])));

tempItem.setX(Integer.parseInt(properties[3]));

tempItem.setY(Integer.parseInt(properties[4]));

tempItem.setFont(new Font(properties[5], Integer.parseInt(properties[6]),

Integer.parseInt(properties[7])));

newString.add(tempItem);

}

in.close();

theString = newString;

}

catch (IOException e)

{

JOptionPane.showMessageDialog(this, "Error: please specify a file to be opened.");

}

}

/\*\*

\* By calling this method, it should

\* get the font chosen by the user from the

\* main window.

\*/

private Font getCurrentFont()

{

String fontName = (String)fontChoices.getSelectedItem();

int fontStyle = fontStyles.getSelectedIndex();

int fontSize = 24;

fontSize = Integer.parseInt((String)fontSizes.getSelectedItem());

Font currentFont = new Font(fontName, fontStyle, fontSize);

return currentFont;

}

}

**Monday, 18/05/2021, 12 – 1:30 pm: Part I of the Learning Journal Exercise:**

/\*\*

\* Contents: This file contains the class DirectoryList with the main method.

\*

\* Author: Dawid Blom.

\*

\* Date: 18/05/2021

\*/

package listing;

import java.io.\*;

import java.util.Scanner;

/\*\*

\* This program will list all the files from

\* the current directory recursively. Meaning,

\* if will list all the files in the current directory,

\* and if the current directory has a sub-directory it will

\* enter the sub-directory and list all the file in that directory.

\* Hence, by doing this recursively we will get all the current directory

\* as well as all the information in the sub-directories as well.

\*/

public class DirectoryList

{

/\*\*

\* By calling this method, it should

\* start the program and continue on to

\* list all the file recursively from the

\* current directory. If the user provides

\* a command-line argument then that argument

\* will be used to specify the current directory,

\* else if the user will be prompt to give a current

\* directory path from which the program will start to

\* run through the files and directories.

\*/

public static void main(String[] args)

{

if (args.length > 0)

{

File path = *commandLineInput*(args[0]);

DirectoryList.*directoryList*(path);

}

else

{

File path = *userInput*();

DirectoryList.*directoryList*(path);

}

}

/\*\*

\* By calling this method, it should recursively

\* traverse the path given by the user and display

\* all the files from the current directory and it's

\* sub-directories.

\*/

public static void directoryList(File path)

{

try

{

File[] fileNames = path.listFiles();

for (File fileName : fileNames)

{

if (fileName.isDirectory())

{

System.***out***.println("Directory: " + fileName.getCanonicalPath());

*directoryList*(fileName);

}

else

System.***out***.println(" Files: " + fileName.getName());

}

}

catch (Exception e)

{

System.***out***.println("");

}

}

/\*\*

\* By calling this method, it should

\* convert the command-line argument

\* that the user provided into a File

\* object.

\*/

public static File commandLineInput(String path)

{

File directory = new File(path);

return directory;

}

/\*\*

\* By calling this method, it should

\* prompt user for input. If the user

\* does no specify the absolute path of the

\* directory they want to list then it will

\* prompt them again.

\*/

public static File userInput()

{

File directory;

System.***out***.println("Enter directory path: ");

while (true)

{

Scanner scanner= new Scanner(System.***in***);

String path = scanner.nextLine().trim();

directory = new File(path);

if (directory.isDirectory() == false && directory.exists() == false)

System.***out***.println("Please enter the full path to the directory: ");

else

break;

}

return directory;

}

}

**Monday, 18/05/2021, 2 – 3 pm: Part II of the Learning Journal Exercise:** I enjoyed coding this program. It made me feel more competent with files and how they work in Java. Furthermore, I do believe I miss understood the discussion for this week, for example, I did not give enough differences between Sockets and Server Sockets. I feel that the course is very high paced in the sense that I would have liked to spend more time on the core concepts of files and how they work. I learned a few concepts about network programming, as well as, some useful file operations and advanced GUI programming. The network programming really surprised me and caused me to wonder since it is a very interesting area, however, I can also see how files are worth the extra attention since most logs will need to be written to a file. This weeks assignment was challenging in the sense that I understand things better when I code it, and this week it was different, all the code was given and we had to implement certain areas of it, therefore, the biggest challenge for me was to first understand how the code works that was given. Moreover, I feel that I am gaining a lot of skills and knowledge in file programming as well as GUI programming. Since, we did not spend a lot of time on the network programming I would like to look into that area a bit more. Furthermore, I realised that I really need to spend more time in the practical side of things. Nonetheless, I am able to apply these new concepts when ever, I need to code a GUI, network server or client, and file operations.