Learning Journal Unit\_8

Dawid Blom.

**Saturday, 29/05/2021, 9 – 11:30 pm: Learning Journal Assignment:** The assignment was good and a good close to the course. The instructions was clear. It truly shows me how scalable Java GUI can be and what all the features are that we can add to it. It was interesting to work with someone else’s code instead of coding everything from scratch. It taught me a lost and also helped me to trust other code, as well as, how to stick to a design pattern given to me.

Code for lab 12. It’s only the three files we were asked to work on. Order of file are: IconSupport.java, GuiDemo.java, TextMenu.java.

package guidemo;

import java.awt.Color;

import java.awt.Cursor;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.event.ActionEvent;

import java.awt.image.BufferedImage;

import java.util.ArrayList;

import javax.swing.AbstractAction;

import javax.swing.Action;

import javax.swing.ImageIcon;

import javax.swing.JMenu;

import javax.swing.JToolBar;

/\*\*

\* Contains a set of Actions that can be used to select images that can

\* be added to a DrawPanel (in the form of ImageItems). Can create a

\* toolbar containing a button for each Action in the set. A button

\* shows an ImageIcon with the image that is selected by that button.

\* Clicking one of the buttons also sets the cursor in the DrawPanel

\* to be a (rough) copy of the image.

\*/

public class IconSupport extends JMenu {

private DrawPanel panel;

private ArrayList<BufferedImage> iconImages = new ArrayList<BufferedImage>();

private ArrayList<Action> actions = new ArrayList<Action>();

public IconSupport(DrawPanel owner) {

panel = owner;

String[] iconNames = {"bell", "camera", "flower", "star", "check", "crossout",

"tux", "bomb", "keyboard","lightbulb", "tv"};

for (String name : iconNames) {

BufferedImage img = Util.getBufferedImageResource("resources/icons/" + name + ".png");

if (img != null) {

iconImages.add(img);

actions.add(new SelectIconAction(name,iconImages.size()-1));

}

}

actions.add(new NoIconAction());

}

/\*\*

\* Return a toolbar containing buttons representing the images that can be added

\* to the DrawPanel.

\* @param horizontal a value of JToolBar.HORIZONTAL or JToolBar.VERTICAL tells

\* whether the toolbar is meant to have horizontal or vertical orientation.

\*/

public JToolBar createToolbar(boolean horizontal) {

JToolBar tbar = new JToolBar( horizontal? JToolBar.HORIZONTAL : JToolBar.VERTICAL);

for (int i = 0; i < actions.size() - 1; i++)

tbar.add(actions.get(i));

tbar.addSeparator(new Dimension(15,0));

tbar.add(actions.get(actions.size()-1));

return tbar;

}

/\*\*

\* By calling this method, it should create menu object

\* containing the same elements found in the createToolBar

\* method at the bottom of the program.

\*

\* Furthermore, all this method does is retrieve

\* the elements in the createToolBar method and

\* loads them into this object that will appear at top

\* of the program.

\*

\*

\* Author: Dawid Blom.

\*/

public JMenu createMenu()

{

JMenu stamper = new JMenu("Stamper");

for (int i = 0; i < actions.size() - 1; i++)

stamper.add(actions.get(i));

stamper.add(actions.get(actions.size() - 1));

return stamper;

}

private class NoIconAction extends AbstractAction {

NoIconAction() {

super("Eraser");

BufferedImage del = new BufferedImage(32,32,BufferedImage.TYPE\_INT\_ARGB);

Graphics g = del.createGraphics();

g.setColor(Color.WHITE);

g.fillRect(0,0,32,32);

g.setColor(Color.RED);

g.drawString("DEL",5,20);

g.dispose();

putValue(Action.SMALL\_ICON, new ImageIcon(del));

putValue(Action.SHORT\_DESCRIPTION, "Use Mouse to Erase Icons"); // tooltip

}

public void actionPerformed(ActionEvent evt) {

panel.setCurrentDrawImage(null);

panel.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR\_CURSOR));

}

}

private class SelectIconAction extends AbstractAction {

int iconNumber;

SelectIconAction(String name, int n) {

// Note: The name is surpressed in toolbars, but not in menus.

super(name,new ImageIcon(iconImages.get(n)));

iconNumber = n;

putValue(Action.SHORT\_DESCRIPTION, "Use Mouse to Stamp this Icon"); // tooltip

}

public void actionPerformed(ActionEvent evt) {

BufferedImage image = iconImages.get(iconNumber);

panel.setCurrentDrawImage(image);

Cursor c = Util.createImageCursor(image, image.getWidth()/2, image.getHeight()/2);

panel.setCursor(c);

}

}

}

package guidemo;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.image.BufferedImage;

import java.io.File;

import javax.imageio.ImageIO;

import javax.swing.\*;

/\*\*

\* A frame that displays a multiline text, possibly with a background image

\* and with added icon images, in a DrawPanel, along with a variety of controlls.

\*/

public class GuiDemo extends JFrame{

/\*\*

\* The main program just creates a GuiDemo frame and makes it visible.

\*/

public static void main(String[] args){

JFrame frame = new GuiDemo();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

private DrawPanel drawPanel;

private SimpleFileChooser fileChooser;

private TextMenu textMenu;

private JCheckBoxMenuItem gradientOverlayCheckbox = new JCheckBoxMenuItem("Gradient Overlay", true);

/\*\*

\* The constructor creates the frame, sizes it, and centers it horizontally on the screen.

\*/

public GuiDemo() {

super("Sayings"); // Specifies the string for the title bar of the window.

JPanel content = new JPanel(); // To hold the content of the window.

content.setBackground(Color.LIGHT\_GRAY);

content.setLayout(new BorderLayout());

setContentPane(content);

// Create the DrawPanel that fills most of the window, and customize it.

drawPanel = new DrawPanel();

drawPanel.getTextItem().setText(

"When the going gets tough the\n" +

"tough keeps going.\n" +

"\nKeep the eyes open and the mind ready."

);

drawPanel.getTextItem().setFontSize(36);

drawPanel.getTextItem().setJustify(TextItem.LEFT);

drawPanel.setBackgroundImage(Util.getImageResource("resources/images/earthrise.jpeg"));

content.add(drawPanel, BorderLayout.CENTER);

// Add an icon toolbar to the SOUTH position of the layout

IconSupport iconSupport = new IconSupport(drawPanel);

content.add( iconSupport.createToolbar(true), BorderLayout.SOUTH );

content.add(makeToolBar(), BorderLayout.NORTH);

// Create the menu bar and add it to the frame. The TextMenu is defined by

// a separate class. The other menus are created in this class.

JMenuBar menuBar = new JMenuBar();

IconSupport menu = new IconSupport(drawPanel);

menuBar.add(menu);

menuBar.add(makeFileMenu());

textMenu = new TextMenu(drawPanel);

menuBar.add(textMenu );

menuBar.add( makeBackgroundMenu() );

setJMenuBar(menuBar);

// Set the size of the window and its position.

pack(); // Size the window to fit its content.

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

setLocation((screenSize.width - getWidth())/2, 50);

// Create and customize the file chooser that is used for file operations.

fileChooser = new SimpleFileChooser();

try { // I'd like to use the Desktop folder as the initial folder in the file chooser.

String userDir = System.getProperty("user.home");

if (userDir != null) {

File desktop = new File(userDir,"Desktop");

if (desktop.isDirectory())

fileChooser.setDefaultDirectory(desktop);

}

}

catch (Exception e) {

}

} // end constructor

/\*\*

\* Create the "File" menu from actions that are defined later in this class.

\*/

private JMenu makeFileMenu() {

JMenu menu = new JMenu("File");

menu.add(newPictureAction);

menu.add(saveImageAction);

menu.addSeparator();

menu.add(quitAction);

return menu;

}

/\*\*

\* Create the "Background" menu, using objects of type ChooseBackgroundAction,

\* a class that is defined later in this file.

\*/

private JMenu makeBackgroundMenu() {

JMenu menu = new JMenu("Background");

menu.add(new ChooseBackgroundAction("Mandelbrot"));

menu.add(new ChooseBackgroundAction("Earthrise"));

menu.add(new ChooseBackgroundAction("Sunset"));

menu.add(new ChooseBackgroundAction("Cloud"));

menu.add(new ChooseBackgroundAction("Eagle\_nebula"));

menu.addSeparator();

menu.add(new ChooseBackgroundAction("Custom..."));

menu.addSeparator();

menu.add(new ChooseBackgroundAction("Color..."));

menu.addSeparator();

menu.add(gradientOverlayCheckbox);

gradientOverlayCheckbox.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent evt) {

if (gradientOverlayCheckbox.isSelected())

drawPanel.setGradientOverlayColor(Color.WHITE);

else

drawPanel.setGradientOverlayColor(null);

}

});

return menu;

}

/\*\*

\* By calling this method, it should

\* create a second tool bar placed underneath the main

\* tool bar with all available back ground content with

\* respect to images, and a new blank sheet.

\*

\* Author: Dawid Blom.

\*/

private JToolBar makeToolBar()

{

JToolBar toolBar = new JToolBar();

toolBar.setBorder(null);

toolBar.add(new ChooseBackgroundAction("Mandelbrot"));

toolBar.add(new ChooseBackgroundAction("Earthrise"));

toolBar.add(new ChooseBackgroundAction("Sunset"));

toolBar.add(new ChooseBackgroundAction("Cloud"));

toolBar.add(new ChooseBackgroundAction("Eagle\_nebula"));

toolBar.add(new ChooseBackgroundAction("Custom..."));

toolBar.add(new ChooseBackgroundAction("Color..."));

return toolBar;

}

private AbstractAction newPictureAction =

new AbstractAction("New", Util.iconFromResource("resources/action\_icons/fileopen.png")) {

public void actionPerformed(ActionEvent evt) {

drawPanel.clear();

gradientOverlayCheckbox.setSelected(true);

textMenu.setDefaults();

}

};

private AbstractAction quitAction =

new AbstractAction("Quit", Util.iconFromResource("resources/action\_icons/exit.png")) {

public void actionPerformed(ActionEvent evt) {

System.exit(0);

}

};

private AbstractAction saveImageAction =

new AbstractAction("Save Image...", Util.iconFromResource("resources/action\_icons/filesave.png")) {

public void actionPerformed(ActionEvent evt) {

File f = fileChooser.getOutputFile(drawPanel, "Select Ouput File", "saying.jpeg");

if (f != null) {

try {

BufferedImage img = drawPanel.copyImage();

String format;

String fileName = f.getName().toLowerCase();

if (fileName.endsWith(".png"))

format = "PNG";

else if (fileName.endsWith(".jpeg") || fileName.endsWith(".jpg"))

format = "JPEG";

else {

JOptionPane.showMessageDialog(drawPanel,

"The output file name must end wth\n.png or .jpeg.");

return;

}

ImageIO.write(img,format,f);

}

catch (Exception e) {

JOptionPane.showMessageDialog(drawPanel, "Sorry, the image could not be saved.");

}

}

}

};

/\*\*

\* An object of type ChooseBackgroudnAction represents an action through which the

\* user selects the background of the picture. There are three types of background:

\* solid color background ("Color..." command), an image selected by the user from

\* the file system ("Custom..." command), and four built-in image resources

\* (Mandelbrot, Earthrise, Sunset, and Eagle\_nebula).

\*/

private class ChooseBackgroundAction extends AbstractAction {

String text;

ChooseBackgroundAction(String text) {

super(text);

this.text = text;

if (!text.equals("Custom...") && !text.equals("Color...")) {

putValue(Action.SMALL\_ICON,

Util.iconFromResource("resources/images/" + text.toLowerCase() + "\_thumbnail.jpeg"));

}

if (text.equals("Color..."))

{

BufferedImage imageIcon = new BufferedImage(32,32,BufferedImage.TYPE\_INT\_RGB);

Graphics g = imageIcon.createGraphics();

g.setColor(Color.GRAY);

g.fillRect(0, 0, 32, 32);

g.setColor(Color.GREEN);

g.fillRect(2, 2, 32, 32);

g.setColor(Color.RED);

g.fillRect(2, 2, 21, 32);

g.setColor(Color.ORANGE);

g.fillRect(2, 2, 11, 32);

putValue(Action.SMALL\_ICON, new ImageIcon(imageIcon));

putValue(Action.SHORT\_DESCRIPTION, "Set solid color instead of image");

}

else if (text.equals("Custom..."))

putValue(Action.SMALL\_ICON, Util.iconFromResource("resources/action\_icons/fileopen.png"));

else

putValue(Action.SHORT\_DESCRIPTION, "Use this image as the background.");

}

public void actionPerformed(ActionEvent evt) {

if (text.equals("Custom...")) {

File inputFile = fileChooser.getInputFile(drawPanel, "Select Background Image");

if (inputFile != null) {

try {

BufferedImage img = ImageIO.read(inputFile);

if (img == null)

throw new Exception();

drawPanel.setBackgroundImage(img);

}

catch (Exception e) {

JOptionPane.showMessageDialog(drawPanel, "Sorry, couldn't read the file.");

}

}

}

else if (text.equals("Color...")) {

Color c = JColorChooser.showDialog(drawPanel, "Select Color for Background", drawPanel.getBackground());

if (c != null) {

drawPanel.setBackground(c);

drawPanel.setBackgroundImage(null);

}

}

else {

Image bg = Util.getImageResource("resources/images/" + text.toLowerCase() + ".jpeg");

drawPanel.setBackgroundImage(bg);

}

}

}

}

package guidemo;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;;

/\*\*

\* A menu full of commands that affect the text shown

\* in a DrawPanel.

\*/

public class TextMenu extends JMenu {

private final DrawPanel panel; // the panel whose text is controlled by this menu

private JRadioButtonMenuItem left;

private JCheckBoxMenuItem bold; // controls whether the text is bold or not.

private JCheckBoxMenuItem italic; // controls whether the text is italic or not.

/\*\*

\* Constructor creates all the menu commands and adds them to the menu.

\* @param owner the panel whose text will be controlled by this menu.

\*/

public TextMenu(DrawPanel owner) {

super("Text");

this.panel = owner;

final JMenuItem change = new JMenuItem("Change Text...");

change.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent evt) {

String currentText = panel.getTextItem().getText();

String newText = GetTextDialog.showDialog(panel,currentText);

if (newText != null && newText.trim().length() > 0) {

panel.getTextItem().setText(newText);

panel.repaint();

}

}

});

/\*\*

\* New feature added to the text. This feature

\* allows the user to select a cutome line spacing

\* between the lines of text.

\*

\* Author: Dawid Blom.

\*/

final JMenuItem lineSpacing = new JMenuItem("Line Spacing...");

lineSpacing.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent evt) {

double currentSize = panel.getTextItem().getLineHeightMultiplier();

String s = JOptionPane.showInputDialog(panel, "Choose your line spacing carefully.",currentSize);

if (s != null && s.trim().length() > 0) {

try {

double newSize = Double.parseDouble(s.trim());

panel.getTextItem().setLineHeightMultiplier(newSize);

panel.repaint();

}

catch (Exception e) {

JOptionPane.showMessageDialog(panel, s + " The line space size must be 1.0 or higher.");

}

}

}

});

final JMenuItem size = new JMenuItem("Set Size...");

size.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent evt) {

int currentSize = panel.getTextItem().getFontSize();

String s = JOptionPane.showInputDialog(panel, "What font size do you want to use?",currentSize);

if (s != null && s.trim().length() > 0) {

try {

int newSize = Integer.parseInt(s.trim()); // can throw NumberFormatException

panel.getTextItem().setFontSize(newSize); // can throw IllegalArgumentException

panel.repaint();

}

catch (Exception e) {

JOptionPane.showMessageDialog(panel, s + " is not a legal text size.\n"

+"Please enter a positive integer.");

}

}

}

});

final JMenuItem color = new JMenuItem("Set Color...");

color.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent evt) {

Color currentColor = panel.getTextItem().getColor();

Color newColor = JColorChooser.showDialog(panel, "Select Text Color", currentColor);

if (newColor != null) {

panel.getTextItem().setColor(newColor);

panel.repaint();

}

}

});

italic = new JCheckBoxMenuItem("Italic");

italic.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent evt) {

panel.getTextItem().setItalic(italic.isSelected());

panel.repaint();

}

});

bold = new JCheckBoxMenuItem("Bold");

bold.addActionListener( new ActionListener() {

public void actionPerformed(ActionEvent evt) {

panel.getTextItem().setBold(bold.isSelected());

panel.repaint();

}

});

add(change);

addSeparator();

add(lineSpacing);

add(size);

add(color);

add(italic);

add(bold);

addSeparator();

add(makeJustifyMenu());

addSeparator();

add(makeFontNameSubmenu());

}

/\*\*

\* Reset the state of the menu to reflect the default settings for text

\* in a DrawPanel. (Sets the italic and bold checkboxes to unselected.)

\* This method is called by the main program when the user selects the

\* "New" command, to make sure that the menu state reflects the contents

\* of the panel.

\*/

public void setDefaults() {

italic.setSelected(false);

bold.setSelected(false);

left.setSelected(true);

}

/\*\*

\* By calling this method, it should create a

\* radio button group that represents the

\* text alignment of the picture. There are

\* three different alignments, Left, Right, and Center.

\* Depending on which one the user clicks, that will be the one

\* that is used.

\*

\* Author: Dawid Blom.

\*/

private JMenu makeJustifyMenu(){

JMenu menu = new JMenu("Justify Text");

ButtonGroup radioBG = new ButtonGroup();

JRadioButtonMenuItem left = new JRadioButtonMenuItem("Left");

ActionListener setLeftJustify = new ActionListener() {

public void actionPerformed(ActionEvent evt) {

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

panel.getTextItem().setJustify(TextItem.LEFT);

panel.repaint();

}

};

left.addActionListener(setLeftJustify);

JRadioButtonMenuItem right = new JRadioButtonMenuItem("Right");

ActionListener setRightJustify = new ActionListener() {

public void actionPerformed(ActionEvent evt) {

panel.getTextItem().setJustify(TextItem.RIGHT);

panel.repaint();

}

};

right.addActionListener(setRightJustify);

JRadioButtonMenuItem center = new JRadioButtonMenuItem("Center");

ActionListener setCenterJustify = new ActionListener() {

public void actionPerformed(ActionEvent evt) {

panel.getTextItem().setJustify(TextItem.CENTER);

panel.repaint();

}

};

center.addActionListener(setCenterJustify);

radioBG.add(left);

radioBG.add(right);

radioBG.add(center);

menu.add(left);

menu.add(right);

menu.add(center);

left.setSelected(true);

return menu;

}

/\*\*

\* Create a menu containing a list of all available fonts.

\* (It turns out this can be very messy, at least on Linux, but

\* it does show the use what is available and lets the user try

\* everything!)

\*/

private JMenu makeFontNameSubmenu() {

ActionListener setFontAction = new ActionListener() {

public void actionPerformed(ActionEvent evt) {

panel.getTextItem().setFontName(evt.getActionCommand());

panel.repaint();

}

};

JMenu menu = new JMenu("Font Name");

String[] basic = { "Serif", "SansSerif", "Monospace" };

for (String f : basic) {

JMenuItem m = new JMenuItem(f+ " Default");

m.setActionCommand(f);

m.addActionListener(setFontAction);

m.setFont(new Font(f,Font.PLAIN,12));

menu.add(m);

}

menu.addSeparator();

String[] fonts = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();

if (fonts.length <= 20) {

for (String f : fonts) {

JMenuItem m = new JMenuItem(f);

m.addActionListener(setFontAction);

m.setFont(new Font(f,Font.PLAIN,12));

menu.add(m);

}

}

else { //Too many items for one menu; divide them into several sub-sub-menus.

char ch1 = 'A';

char ch2 = 'A';

JMenu m = new JMenu();

int i = 0;

while (i < fonts.length) {

while (i < fonts.length && (Character.toUpperCase(fonts[i].charAt(0)) <= ch2 || ch2 == 'Z')) {

JMenuItem item = new JMenuItem(fonts[i]);

item.addActionListener(setFontAction);

item.setFont(new Font(fonts[i],Font.PLAIN,12));

m.add(item);

i++;

}

if (i == fonts.length || (m.getMenuComponentCount() >= 12 && i < fonts.length-4)) {

if (ch1 == ch2)

m.setText("" + ch1);

else

m.setText(ch1 + " to " + ch2);

menu.add(m);

if (i < fonts.length)

m = new JMenu();

ch2++;

ch1 = ch2;

}

else

ch2++;

}

}

return menu;

}

}