//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Author: Patrick Austin

// Date: 12/4/2015

// Project: CS 326 Homework 8 - Color Sampler in Java

// Note: !!!Requires correctly formatted "colors.txt" as per

// the prompt to run!!!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//import needed packages

import java.awt.\*;

import javax.swing.\*;

import java.awt.event.\*;

import javax.swing.event.\*;

import java.io.\*;

//ColorData structure is used to hold initial readin data for color values,

//and is maintained when they are altered.

class ColorData

{

public String colorName;

public int redValue;

public int greenValue;

public int blueValue;

}

//ColorSampler class holds necessary data/variables for program operation

public class ColorSampler extends JFrame

{

//declare members

//array of colorData objects, loaded from "colors.txt"

protected ColorData data[];

//red blue and green ints to store user's current 'active' color values

protected int currentRed;

protected int currentBlue;

protected int currentGreen;

//int index for current color option

protected int index;

//string to hold name of the GUI window

protected String windowTitle;

//color box object, to view the chosen color in GUI

protected ColorBox colorWindow;

//red blue and green labels for GUI

protected JLabel redLabel;

protected JLabel greenLabel;

protected JLabel blueLabel;

//red blue and green fields for displaying current value

protected JTextField redField;

protected JTextField greenField;

protected JTextField blueField;

//red blue and green buttons to modify the current value, + and -

protected JButton redMinus;

protected JButton redPlus;

protected JButton greenMinus;

protected JButton greenPlus;

protected JButton blueMinus;

protected JButton bluePlus;

//save button to overwrite saved value with current one

protected JButton saveButton;

//reset button to restore color to saved value

protected JButton resetButton;

//list of colors for JList element

protected JList colorList;

//main function attempts to create a ColorSampler object, which will trigger all

//program operation. Catches IO exception if ColorSampler constructor throws one.

public static void main(String argv [])

{

try

{

new ColorSampler("Color Sampler");

}

catch (IOException noFile)

{

System.out.println("ERROR: file colors.txt not found, aborting program.");

}

}

//ColorSampler constructor does file readin, initializes and sets up GUI objects,

//and creates the listeners that will conduct subsequent operations

public ColorSampler(String title) throws IOException

{

//initilize data readin

//open stream, reader, tokenizer

FileInputStream stream = new FileInputStream("colors.txt");

InputStreamReader reader = new InputStreamReader(stream);

StreamTokenizer tokens = new StreamTokenizer(reader);

//allocate ColorData objects to store read in data

//!!! ASSUMES FILE FORMATTED AS PER PROMPT !!!

//if more or fewer colors than in prompt are expected, these numbers

//will need to be modified accordingly.

data = new ColorData[11];

//allocate seperate list of color names for JList operation

String nameList[] = new String[11];

//Conduct data readin- loop for each expected color, reading in each member.

//Copy each color name into nameList as well.

for ( int i = 0; i < 11; i++ )

{

//create new ColorData object at index

data[i] = new ColorData();

//read in one color name and three ints, r/g/b

tokens.nextToken();

data[i].colorName = (String) tokens.sval;

tokens.nextToken();

data[i].redValue = (int) tokens.nval;

tokens.nextToken();

data[i].greenValue = (int) tokens.nval;

tokens.nextToken();

data[i].blueValue = (int) tokens.nval;

//copy color name into nameList

nameList[i] = data[i].colorName;

}

stream.close();

//initialize GUI window

//set up base window

//set window bounds

setBounds(100,100,370,370);

//set window title

windowTitle = title;

//create listener for when window is closed

addWindowListener(new WindowDestroyer());

//set content pane to null

getContentPane().setLayout(null);

//set up color window for displaying the current color

//create the object- uses current ColorSampler in constructor

colorWindow = new ColorBox(this);

//program will default to showing first color on startup- make index 0

index = 0;

//give the color window the current R/G/B values

colorWindow.red = data[index].redValue;

colorWindow.green = data[index].greenValue;

colorWindow.blue = data[index].blueValue;

//make the starting R/G/B values the current values

currentRed = data[index].redValue;

currentGreen = data[index].greenValue;

currentBlue = data[index].blueValue;

//add the window to the content pane

getContentPane().add(colorWindow);

//set bounds for the color window

colorWindow.setBounds(10, 10, 220, 150);

//set up GUI element for selectable list of colors

//create object- use nameList array for constructor

colorList = new JList<>(nameList);

//add to content pane

getContentPane().add(colorList);

//set bounds

colorList.setBounds(245, 10, 110, 330);

//set the selected item in the list to the one at index

colorList.setSelectedIndex(index);

//add listener for user to change the selection in the list

colorList.addListSelectionListener(new ListHandler());

//set up red label GUI element

//create object

redLabel = new JLabel("Red:");

//add to content pane

getContentPane().add(redLabel);

//set bounds

redLabel.setBounds(10, 170, 60, 30);

//set up green label GUI element - as above

greenLabel = new JLabel("Green:");

getContentPane().add(greenLabel);

greenLabel.setBounds(10, 210, 60, 30);

//set up blue label GUI element - as above

blueLabel = new JLabel("Blue:");

getContentPane().add(blueLabel);

blueLabel.setBounds(10, 250, 60, 30);

//set up red field GUI element, displaying current red

//create object

redField = new JTextField(Integer.toString(currentRed));

//add to content pane

getContentPane().add(redField);

//set bounds

redField.setBounds(70, 170, 50, 30);

//set uneditable- user cannot modify field

redField.setEditable(false);

//set up green field GUI element as above

greenField = new JTextField(Integer.toString(currentGreen));

getContentPane().add(greenField);

greenField.setBounds(70, 210, 50, 30);

greenField.setEditable(false);

//set up blue field GUI element as above

blueField = new JTextField(Integer.toString(currentBlue));

getContentPane().add(blueField);

blueField.setBounds(70, 250, 50, 30);

blueField.setEditable(false);

//set up red minus button element

//create object

redMinus = new JButton("-");

//add to content pane

getContentPane().add(redMinus);

//set bounds

redMinus.setBounds(130, 170, 45, 30);

//create listener for user interaction with button

redMinus.addActionListener(new ActionHandler());

//set up red plus button element, as above

redPlus = new JButton("+");

getContentPane().add(redPlus);

redPlus.setBounds(185, 170, 45, 30);

redPlus.addActionListener(new ActionHandler());

//set up green minus button element, as above

greenMinus = new JButton("-");

getContentPane().add(greenMinus);

greenMinus.setBounds(130, 210, 45, 30);

greenMinus.addActionListener(new ActionHandler());

//set up green plus button element, as above

greenPlus = new JButton("+");

getContentPane().add(greenPlus);

greenPlus.setBounds(185, 210, 45, 30);

greenPlus.addActionListener(new ActionHandler());

//set up blue minus button element, as above

blueMinus = new JButton("-");

getContentPane().add(blueMinus);

blueMinus.setBounds(130, 250, 45, 30);

blueMinus.addActionListener(new ActionHandler());

//set up blue plus button element, as above

bluePlus = new JButton("+");

getContentPane().add(bluePlus);

bluePlus.setBounds(185, 250, 45, 30);

bluePlus.addActionListener(new ActionHandler());

//set up save button element

//create object

saveButton = new JButton("Save");

//add to content pane

getContentPane().add(saveButton);

//set bounds

saveButton.setBounds(30, 300, 80, 30);

//add listener for user interaction with button

saveButton.addActionListener(new ActionHandler());

//set up reset button element

//create object

resetButton = new JButton("Reset");

//add to content pane

getContentPane().add(resetButton);

//set bounds

resetButton.setBounds(120, 300, 80, 30);

//add listener for user interaction with button

resetButton.addActionListener(new ActionHandler());

//GUI intiailization complete- make window visible

setVisible(true);

}

//ListHandler manages case where user selects a new color from the JList.

//Must update current color to saved color from the new selection, update GUI,

//discard \* in title bar if there is one, and send/repaint new color in the

//ColorBox.

private class ListHandler implements ListSelectionListener

{

public void valueChanged(ListSelectionEvent event)

{

if ( !event.getValueIsAdjusting() )

{

setTitle(windowTitle);

index = colorList.getSelectedIndex();

currentRed = data[index].redValue;

currentGreen = data[index].greenValue;

currentBlue = data[index].blueValue;

colorWindow.red = currentRed;

colorWindow.green = currentGreen;

colorWindow.blue = currentBlue;

redField.setText(Integer.toString(currentRed));

greenField.setText(Integer.toString(currentGreen));

blueField.setText(Integer.toString(currentBlue));

repaint();

}

}

}

//ActionHandler manages interaction with the various GUI buttons

private class ActionHandler implements ActionListener

{

//if a button is pressed...

public void actionPerformed(ActionEvent event)

{

//for a - button: if in bounds, decrement current value, update value in

//GUI, send new value to the ColorBox, add an \* to the title bar, and

//repaint the color

if ( event.getSource() == redMinus )

{

if (currentRed >= 5)

{

setTitle(windowTitle+"\*");

currentRed -= 5;

colorWindow.red = currentRed;

redField.setText(Integer.toString(currentRed));

repaint();

}

}

//for a + button: if in bounds, increment current value, update value in

//GUI, send new value to the ColorBox, add an \* to the title bar, and

//repaint the color

if ( event.getSource() == redPlus )

{

if (currentRed <= 250)

{

setTitle(windowTitle+"\*");

currentRed += 5;

colorWindow.red = currentRed;

redField.setText(Integer.toString(currentRed));

repaint();

}

}

//for the other - and + buttons, proceed as above

if ( event.getSource() == greenMinus )

{

if (currentGreen >= 5)

{

setTitle(windowTitle+"\*");

currentGreen -= 5;

colorWindow.green = currentGreen;

greenField.setText(Integer.toString(currentGreen));

repaint();

}

}

if ( event.getSource() == greenPlus )

{

if (currentGreen <= 250)

{

setTitle(windowTitle+"\*");

currentGreen += 5;

colorWindow.green = currentGreen;

greenField.setText(Integer.toString(currentGreen));

repaint();

}

}

if ( event.getSource() == blueMinus )

{

if (currentBlue >= 5)

{

setTitle(windowTitle+"\*");

currentBlue -= 5;

colorWindow.blue = currentBlue;

blueField.setText(Integer.toString(currentBlue));

repaint();

}

}

if ( event.getSource() == bluePlus )

{

if (currentBlue <= 250)

{

setTitle(windowTitle+"\*");

currentBlue += 5;

colorWindow.blue = currentBlue;

blueField.setText(Integer.toString(currentBlue));

repaint();

}

}

//For the reset button, restore saved values, send restored values to the

//colorBox, update GUI, set title bar to no \*, and repaint color

if ( event.getSource() == resetButton )

{

setTitle(windowTitle);

currentRed = data[index].redValue;

currentGreen = data[index].greenValue;

currentBlue = data[index].blueValue;

colorWindow.red = currentRed;

colorWindow.green = currentGreen;

colorWindow.blue = currentBlue;

redField.setText(Integer.toString(currentRed));

greenField.setText(Integer.toString(currentGreen));

blueField.setText(Integer.toString(currentBlue));

repaint();

}

//For the save button, set title bar to no \* and replace saved values with

//current values

if ( event.getSource() == saveButton )

{

setTitle(windowTitle);

data[index].redValue = currentRed;

data[index].greenValue = currentGreen;

data[index].blueValue = currentBlue;

}

}

}

//WindowDestroyer manages interaction when user closes GUI window- saves color

//data back to colors.txt file. Throws IOException and exits if file write fails.

private class WindowDestroyer extends WindowAdapter

{

public void windowClosing(WindowEvent closing)

{

try

{

FileOutputStream stream = new FileOutputStream("colors.txt");

PrintWriter writer = new PrintWriter(stream);

for ( int i = 0; i < data.length; i++ )

{

writer.println( data[i].colorName + "\t" + data[i].redValue + "\t"

+ data[i].greenValue + "\t" + data[i].blueValue);

}

writer.flush();

stream.close();

}

catch (IOException writeError)

{

System.out.println("ERROR: save to colors.txt failed.");

System.exit(0);

}

System.exit(0);

}

}

}

//ColorBox class handles paint operation when color values change.

class ColorBox extends JComponent

{

//ColorSampler object for accessing dimension data

ColorSampler parent;

//ints R/G/B hold the info for the color to be painted

int red;

int green;

int blue;

//constructor- ColorBox has a ColorSampler member to get at needed data

public ColorBox(ColorSampler a)

{

parent = a;

}

//paint function- uses r/g/b members and info about pane dimentions to paint

//the box with the target color

public void paint(Graphics box)

{

Dimension dim = getSize();

box.setColor(new Color(red/255f,green/255f,blue/255f));

box.fillRect(1, 1, dim.width-2, dim.height-2);

}

}