Лабораторная работа № 7

Цель:

Научиться использовать существующие библиотеки для реализации собственных проектов.

Постановка задачи:

Задание 1. Программа работы с изображением. Нужно загружать картинку из файла и разбивать ее на части (например, на равные прямоугольники, с помощью класса PixelGrabber). Затем из этих частей нужно собрать целую картинку, например, по принципу «пятнашек» или перетягиванием мыши. Можно предложить собственный вариант сборки. При удачной сборке картинки нужно выдавать соответствующее сообщение. Образец сборки должен быть доступен.

Задание 2. Программа Mini Word Art. Изменяя текст и цвет, формируется объемный текст. Цвета объема просчитывать, используя положение источника света (минимум с двух позиций, например, слева и справа или т.п.).

Более предпочтительным является использование существующих решений.

Решение задачи:

Задание 1.

Загрузку изображения из файла осуществим с помощью объекта класса “JFileChooser”, к которому будем обращаться при нажатии на вкладку “JMenu” “file → open ”. Также установим фильтры с расширениями “.png” и “.jpg”. После того как файл выбран, считываем изображение и запоминаем его высоту и ширину. Далее на каждой итерации двойного цикла будем делить изображение на фрагменты одинакового размера. В двумерный массив “JLabel[][]” заносим фрагменты в качестве иконок. Необходимо создать для каждого фрагмента новый объект класса “Fragment”, поля которого позволят запомнить для каждой иконки ее правильное местоположение относительно общей картины.

```
JLabel[][] labels = new JLabel[n][n];
JPanel labelPanel;
fragments = new ArrayList<>();
ArrayList<Icon> icons = new ArrayList<>();
labelPanel = new MyPanel(labels);
Timer t2 = new Timer();

open.addActionListener(e -> {
    JFileChooser fileChooser = new JFileChooser( currentDirectoryPath: ".");
    fileChooser.setAcceptAllFileFilterUsed(false);
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    fileChooser.addChoosableFileFilter(new FileNameExtensionFilter( description: "Файл \"JPG\"",  ...extensions: "jpg"));
    fileChooser.addChoosableFileFilter(new FileNameExtensionFilter( description: "Файл \"PNG\"",  ...extensions: "png"));
    if (fileChooser.showOpenDialog( parent: this) == JFileChooser.APPROVE_OPTION) {
        try {
            image = ImageIO.read(new File(fileChooser.getSelectedFile().getPath()));
            panel.setSize(image.getWidth(), image.getHeight());
            setSize( width: image.getWidth()+50,  height: image.getHeight()+70);
            setResizable(false);
            win = ImageIO.read(new File( pathname: "C:\\Users\\Лиза\\IdeaProjects\\Practice7.1\\src\\winner.png"));
            int h = image.getHeight();
            int w = image.getWidth();
            icons.clear();
            fragments.clear();
            for(int i = 0; i < n; i++){
                for(int j = 0; j < n; j++){
                    labels[i][j].setIcon(new ImageIcon(image.getSubimage( x: j*(w/n),  y: i*(h/n),  w: w/n,  h: h/n)));
                    labels[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
                    icons.add(labels[i][j].getIcon());
                    fragments.add(new Fragment(labels[i][j], i, j));
                }
            }
        } catch (IOException exc) {
            exc.printStackTrace();
        }
```

Рис.1. Загрузка изображения из файла. Создание фрагментов картинки.

```
class Fragment{
    private Icon icon;
    private int i, j;

    Fragment(JLabel label, int i, int j){
        this.icon = label.getIcon();
        this.i = i;
        this.j = j;
    }

    boolean getRightPlace(int i, int j, Icon icon){
        return this.i == i && this.j == j && this.icon == icon;
    }

}
```

Рис.2. Класс “Fragment”.

Следующим шагом является запуск таймера. Спустя 3 секунды после загрузки изображения, иконки случайным образом занимают новые места. (Метод выбора рандомного местоположения будет описан позже). Второй таймер нужен для проверки правильности сборки картины: если булева функция “win” возвращает “true”, то всплывает диалоговое окно, символизирующее победу, а таймер прекращает свою работу.

```
        Timer t1 = new Timer();
        TimerTask timerTask = () -> {
                ArrayList<Integer> array = getRandomNumber();
                int f =0;
                for(int i = 0; i < n; i++){
                    for(int j = 0; j < n; j++){
                        labels[i][j].setIcon(icons.get(array.get(f)));
                        f++;
                    }
                }
        };
        show.setEnabled(true);
        t1.schedule(timerTask,  delay: 3000);

        TimerTask timerTask2 = () -> {
                if(win(fragments, labels)){
                    JPanel winPanel = new JPanel(new BorderLayout());
                    JLabel winner = new JLabel(new ImageIcon(win));
                    winPanel.add(winner, BorderLayout.CENTER);

                    JOptionPane.showMessageDialog(panel, winPanel,
                            title: "Congratulations!", JOptionPane.PLAIN_MESSAGE ,  icon: null);
                    cancel();
                }
        };
        t2.schedule(timerTask2,  delay: 6000,  period: 100);
    }
});
```

Рис.3.Таймеры и их задания.

Метод “getRandomNumber()” заполняет случайными числами массив размерности по количеству иконок. Таким образом, объекту “labels[i][j]” устанавливается иконка из списка “icons”, номер которой определяет элемент массива, возвращаемого данным методом.

```
private ArrayList<Integer> getRandomNumber() {
    ArrayList<Integer> numbersGenerated = new ArrayList<>();

    for (int i = 0; i < Math.pow(n, 2); i++) {
        Random randNumber = new Random();
        int iNumber = randNumber.nextInt((int)Math.pow(n, 2));

        if (!numbersGenerated.contains(iNumber)) {
            numbersGenerated.add(iNumber);
        } else {
            i--;
        }
    }
    return numbersGenerated;
}
```

Рис.4. Реализация метода “getRandomNumber()”.

Функция “win”, как было сказано ранее, определяет, на своем ли месте находится иконка: для каждого фрагмента вызывается метод “getRightPlace()”, который возвращает “false”, если текущая позиция иконки не совпадает с ее позицией при создании объекта.

```
private boolean win(ArrayList<Fragment> fragments, JLabel[][]labels){
    int f =0;
    for (int i =0; i < n; i++){
        for(int j =0; j < n; j++){
            if(!(fragments.get(f).getRightPlace(i, j, labels[i][j].getIcon()))){
                return false;
            }
            f++;
        }
    }
    return true;
}
```

Рис.5.Метод “win”.

Слушатель панели меню “Help → Show image” при нажатии отображает в диалоговом окне подсказку - картинку в собранном виде.

```
show.addActionListener(e -> {

    JLabel helpLabel = new JLabel(new ImageIcon(image));
    JPanel inputPanel = new JPanel();
    inputPanel.add(helpLabel);
    JOptionPane.showMessageDialog( parentComponent: this, inputPanel,
            title: "Help", JOptionPane.PLAIN_MESSAGE);
});

exit.addActionListener(e -> {
    setVisible(false);
    dispose();
});
```

Рис.6. Слушатели для вкладок “show” и “exit”.

В классе “MyPanel”, реализующем интерфейсы “MouseListener”, “MouseMotionListener”, выполняем обработку нажатия мыши. Если координаты клика находятся в зоне какого-либо фрагмента, то заносим его позицию в “Arraylist<Change>”. Если в “components” содержится ровно два объекта, меняем их иконки местами, а сам массив очищаем. Объекты класса “Change” хранят иконку и ее текущую позицию на экране.

```
class Change{
    private int i, j;
    private JLabel component;
    Change(JLabel component, int i, int j){
        this.component = component;
        this.i = i;
        this.j = j;
    }
    JLabel getComponent() { return component; }
    int getI() { return i; }
    int getJ() { return j; }
}
```

Рис.7.Класс “Change”.

```
public void mouseClicked(MouseEvent e) {
    if(e.getX() > 0 && e.getX() < getWidth()/3){
        if(e.getY() > 0 && e.getY() < this.getHeight()/3){
            components.add(new Change(labels[0][0], i: 0, j: 0)); }
        else if(e.getY() > this.getHeight()/3 && e.getY() < 2*this.getHeight()/3){
            components.add(new Change(labels[1][0], i: 1, j: 0)); }
        else if(e.getY() > 2*this.getHeight()/3 && e.getY() < this.getHeight()){
            components.add(new Change(labels[2][0], i: 2, j: 0)); }
    }
    else if(e.getX()< 2*this.getWidth()/3 && e.getX()>this.getWidth()/3){
        if(e.getY() > 0 && e.getY() < this.getHeight()/3){
            components.add(new Change(labels[0][1], i: 0, j: 1)); }
        else if(e.getY() > this.getHeight()/3 && e.getY() < 2*this.getHeight()/3){
            components.add(new Change(labels[1][1], i: 1, j: 1)); }
        else if(e.getY() > 2*this.getHeight()/3 && e.getY() < this.getHeight()){
            components.add(new Change(labels[2][1], i: 2, j: 1)); }
    }
    else if(e.getX()> 2*this.getWidth()/3 && e.getX()<this.getWidth()){
        if(e.getY() > 0 && e.getY() < getHeight()/3){
            components.add(new Change(labels[0][2], i: 0, j: 2)); }
        else if(e.getY() > this.getHeight()/3 && e.getY() < 2*this.getHeight()/3){
            components.add(new Change(labels[1][2], i: 1, j: 2)); }
        else if(e.getY() > 2*this.getHeight()/3 && e.getY() < this.getHeight()){
            components.add(new Change(labels[2][2], i: 2, j: 2)); }
    }

    if(components.size()==2){
        int iF = components.get(0).getI();
        int iS = components.get(1).getI();
        int jF = components.get(0).getJ();
        int jS = components.get(1).getJ();
        Icon temp0 =components.get(0).getComponent().getIcon();
        Icon temp1 =components.get(1).getComponent().getIcon();
        labels[iF][jF].setIcon(temp1);
        labels[iS][jS].setIcon(temp0);
        components.clear();
    }
}
```

Рис.8.Обработка нажатия левой кнопки мыши.

Задание 2.

Сперва следует подключить библиотеку “Text3DMesh”, в которой уже реализованы классы для создания и работы с 3D текстом.

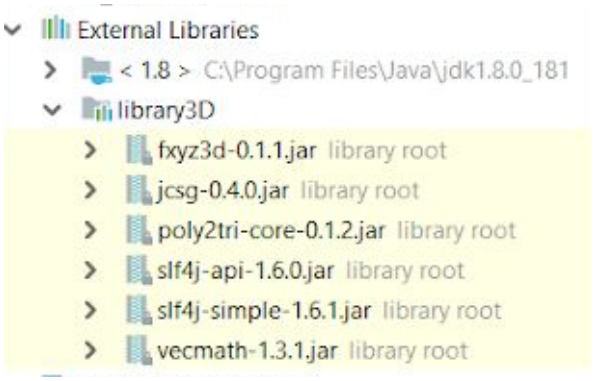

Рис.9.Подключенные библиотеки для работы с 3D текстом.

Для изменения цвета текста воспользуемся объектом класса “ComboBox”, темы которого зададим названиями оттенков.

```
final List<ColorItem> list = new ArrayList<>();
list.add(new ColorItem( name: "Snow", Color.SNOW));
list.add(new ColorItem( name: "DarkCyan", Color.DARKCYAN));
list.add(new ColorItem( name: "Purple", Color.PURPLE));
list.add(new ColorItem( name: "Yellow", Color.YELLOW));
list.add(new ColorItem( name: "DeepSkyBlue", Color.DEEPSKYBLUE));
list.add(new ColorItem( name: "Aquamarine", Color.AQUAMARINE));
list.add(new ColorItem( name: "Crimson", Color.CRIMSON));
list.add(new ColorItem( name: "Black", Color.BLACK));
list.add(new ColorItem( name: "Indigo", Color.INDIGO));
list.add(new ColorItem( name: "Aqua", Color.AQUA));
final List<String> colors = new ArrayList<>();
for(ColorItem colorItem : list){
    colors.add(colorItem.getName());
}
ObservableList<String> items = FXCollections.observableArrayList(colors);
ComboBox<String> comboBox = new ComboBox<>(items);
comboBox.setValue(items.get(6));
TextField textField = new TextField();
VBox settings = new VBox( spacing: 20);
```

Рис.10.Заполнение “comboBox”.

Регулировку эффекта тени “DropShadow” будем выполнять с помощью ползунков, изменение значений которых будет влиять на положение тени относительно текста.

```
Slider sliderHeight = new Slider();
Slider sliderWidth = new Slider();
sliderHeight.setMin(-50);
sliderHeight.setMax(50);
sliderWidth.setMin(-50);
sliderWidth.setMax(50);
sliderHeight.setValue(-50);
sliderWidth.setValue(50);
sliderHeight.setShowTickMarks(true);
sliderWidth.setShowTickMarks(true);
DropShadow dropShadow = new DropShadow();
sliderHeight.valueProperty().addListener((obsVal, oldVal, newVal) -> {
    yValue = (-1) *newVal.intValue();
    dropShadow.setOffsetY(yValue);
});
sliderWidth.valueProperty().addListener((obsVal, oldVal, newVal) -> {
    xValue = newVal.intValue();
    dropShadow.setOffsetX(xValue);
});
dropShadow.setOffsetY(yValue);
dropShadow.setOffsetX(xValue);
```

Рис.11.Эффект отбрасываемой текстом тени и его регулировка.

Для того чтобы текст можно было вращать относительно определенной точки, создаем камеру, задаем ей координаты и добавляем ее на “subScene”,

```
Camera camera = new PerspectiveCamera( fixedEyeAtCameraZero: true);
CameraTransformer cameraTransform = new CameraTransformer();
cameraTransform.setTranslate( x: 0,  y: 0,  z: 0);
cameraTransform.getChildren().add(camera);
camera.setNearClip(0.1);
camera.setFarClip(10000.0);
camera.setTranslateX(1300);
camera.setTranslateZ(-3000);
cameraTransform.rx.setAngle(10.0);
Group group = new Group(cameraTransform);
SubScene subScene = new SubScene(group,  width: 1300, HEIGHT);
BorderPane root = new BorderPane();
root.setCenter(subScene);
root.setLeft(settings);
Scene scene = new Scene(root,  width: WIDTH+600, HEIGHT);
cameraTransform.getChildren().add(new AmbientLight());
subScene.setCamera(camera);
```

Рис.12.Установка камеры.

Само вращение реализуем с помощью задания угла относительно зажимания левой кнопки мыши и передвижения курсора в таком положении.

```
subScene.setOnMousePressed((MouseEvent me) -> {
    mousePosX = me.getSceneX();
    mousePosY = me.getSceneY();
    mouseOldX = me.getSceneX();
    mouseOldY = me.getSceneY();
});
subScene.setOnMouseDragged((MouseEvent me) -> {
    sliderHeight.setValue(0);
    sliderWidth.setValue(0);
    mouseOldX = mousePosX;
    mouseOldY = mousePosY;
    mousePosX = me.getSceneX();
    mousePosY = me.getSceneY();
    mouseDeltaX = (mousePosX - mouseOldX);
    mouseDeltaY = (mousePosY - mouseOldY);
    double modifier = 10.0;
    double modifierFactor = 0.1;
    if (me.isControlDown()) { modifier = 0.1; }
    if (me.isShiftDown()) { modifier = 50.0; }
    if (me.isPrimaryButtonDown()) {
        cameraTransform.ry.setAngle(((cameraTransform.ry.getAngle() +
                mouseDeltaX * modifierFactor * modifier * 2.0) % 360 + 540) % 360 - 180);
        cameraTransform.rx.setAngle(((cameraTransform.rx.getAngle() -
                mouseDeltaY * modifierFactor * modifier * 2.0) % 360 + 540) % 360 - 180);
    } else if (me.isSecondaryButtonDown()) {
        double z = camera.getTranslateZ();
        double newZ = z + mouseDeltaX * modifierFactor * modifier;
        camera.setTranslateZ(newZ);
    } else if (me.isMiddleButtonDown()) {
        cameraTransform.t.setX(cameraTransform.t.getX() +
                mouseDeltaX * modifierFactor * modifier * 0.3);
        cameraTransform.t.setY(cameraTransform.t.getY() +
                mouseDeltaY * modifierFactor * modifier * 0.3);
    }
});
```

Рис.13.Слушатель для перетаскивания курсора при зажимании левой кнопки мыши.

При нажатии кнопки “ok” из “inputPanel” считываем текст, введенный пользователем, и создаем объект класса “Text3DMesh”, используя один из стандартных конструкторов с параметрами. Добавляем эффект тени и цвет согласно параметрам “sliderHeight”(“sliderWidth”) и “comboBox” соответственно.

```
ok.setOnAction(event -> {
    group.getChildren().clear();
    letters = new Text3DMesh( text3D: "   " +textField.getText(),  font: "Gadugi Bold",
            fontSize: 180,  joinSegments: true,  height: 100,  gap: 10,  level: 1);
    letters.setTextureModeNone(getColorItem(comboBox.getValue(), list));

    letters.setEffect(dropShadow);
    group.getChildren().addAll(letters);

    IntStream.range(0, letters.getChildren().size()).
            forEach(i -> ((TexturedMesh) (letters.getChildren().get(i))).getTranslate().setY(100d ));

});
```

Рис.14.Создание 3D текста после нажатия кнопки “ok”.

Класс “ColorItem” нужен для хранения цвета в качестве объекта класса “Color” и его названия.

```
class ColorItem{
    private String name;
    private Color color;

    ColorItem(String name, Color color){
        this.name = name;
        this.color = color;
    }

    Color getColor() { return color; }
    String getName() { return name; }

}
```

Рис.15.Класс “ColorItem”.

Чтобы облегчить работу с передачей темы “comboBox” в конструктор “Text3DMesh”, создадим метод “getColorItem()”, возвращающий определенный цвет, если название такового было выбрано и нашлось в списке.

```
private Color getColorItem(String name, List<ColorItem> list){
    for(ColorItem item : list){
        if(name.equals(item.getName())){ return item.getColor(); }
    }
    return null;
}
```

Рис.16.Метод "getColorItem()" для передачи цвета в конструктор создания 3D текста.

Результат:

Рис.17.Вид оконного приложения после запуска.

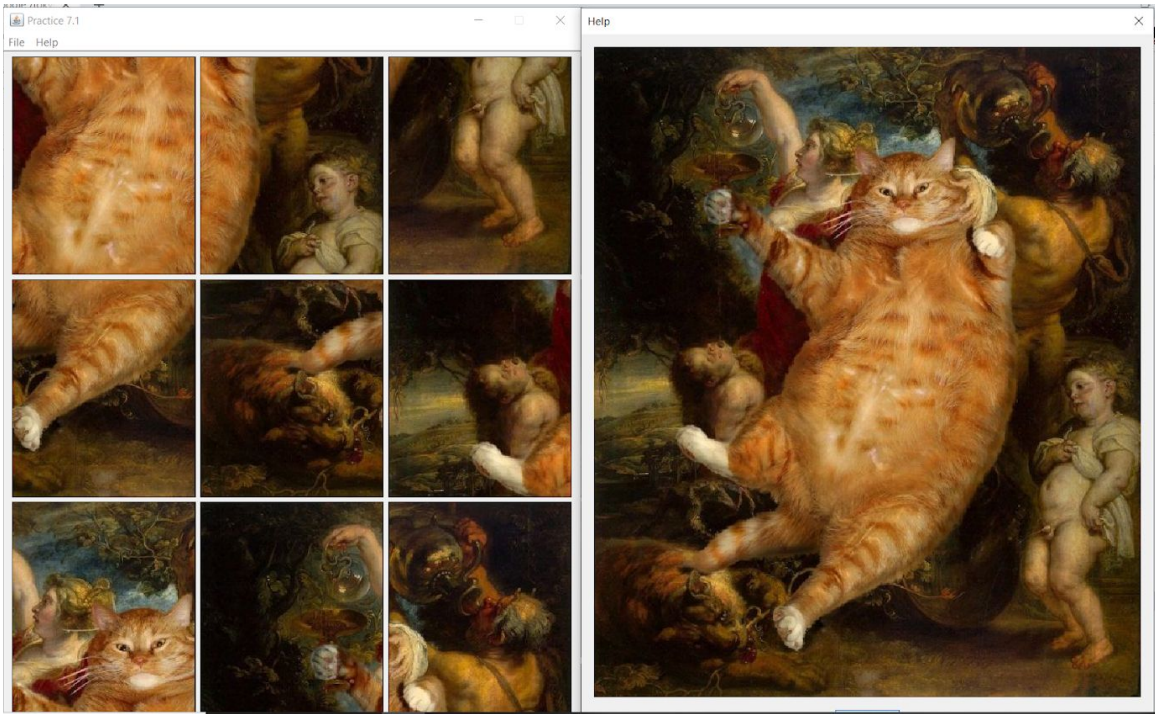

Рис.18.Распределенные случайным образом фрагменты загруженного изображения (слева) и диалоговое окно с подсказкой (справа).

Рис.19.Результат работы программы после удачной сборки картинки.

Рис.20.Результат работы программы.

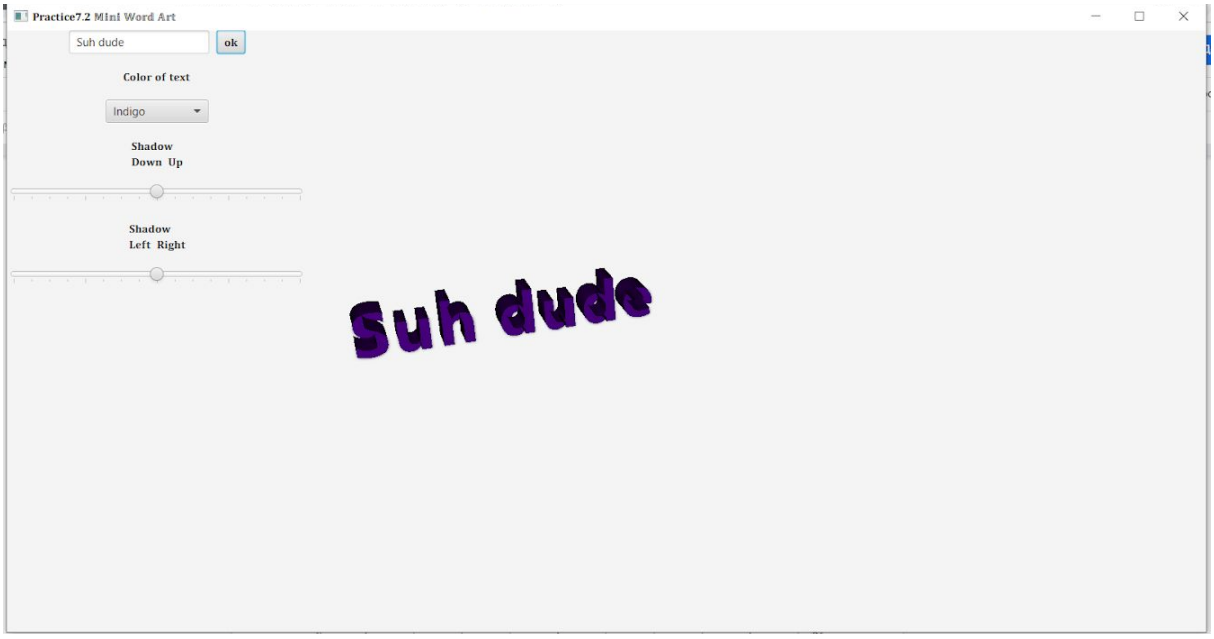

Рис.21.Изменение цвета в панели настроек (“comboBox”).

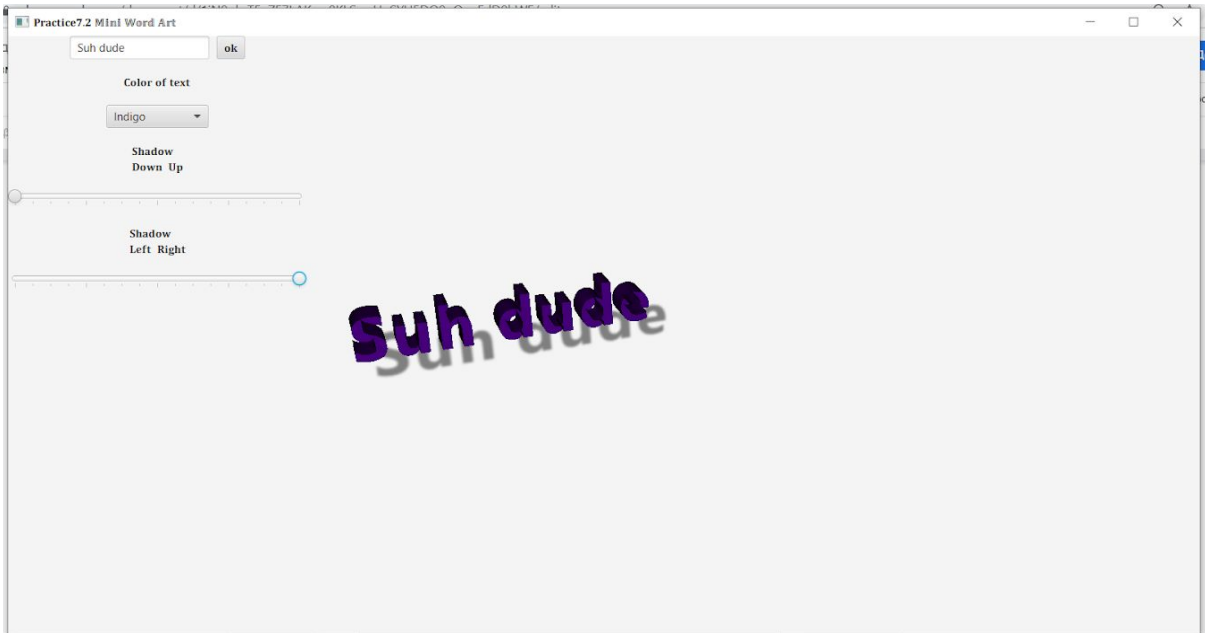

Рис.22.Добавление эффекта отбрасываемой тени (перетаскивание ползунков в панели настроек).

Рис.23.Демонстрация вращения текста.

Рис.24.Демонстрация вращения текста.

Рис.25.Демонстрация вращения текста с изменением оттенка объема.