Лабораторная работа 2

Работа с изображениями

Теоретическая часть

1. Разработка интерфейса

Создайте новый проект JavaFX. Для размещения элементов будем использовать корневой контейнер AnchorPane. На вкладке Layout установите значение характеристикам Pref Width и Pref Height - 600 и 300 соответственно.

На вкладке Hierarchy в компонент AnchorPane добавьте новый компонент SplitPane (horizontal). Кликните по нему правой кнопкой мыши и выберите Fit to Parent.

На левой вкладке необходимо разместить 5 компонентов Slider. Над каждым из них разместить надписи, поясняющие назначение компонента. Ниже разместить три кнопки. Для более удобного расположения также можно воспользоваться контейнером, например VBox.

В правой вкладе разместим компонент ImageView.

Пример размещения компонентов показан на рисунке 1, а расположение компонентов на рисунке 2.

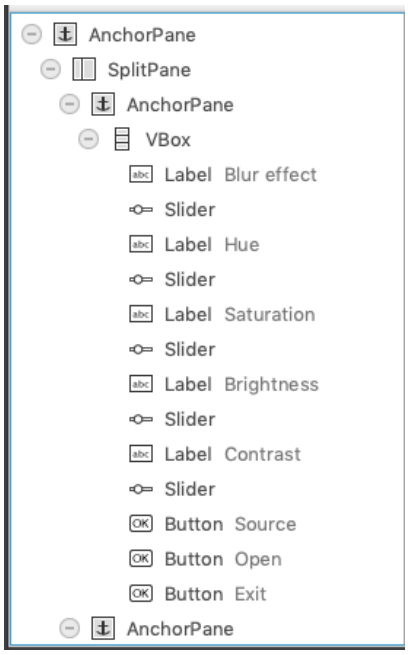

Рисунок 1 – Иерархия контейнеров и компонентов

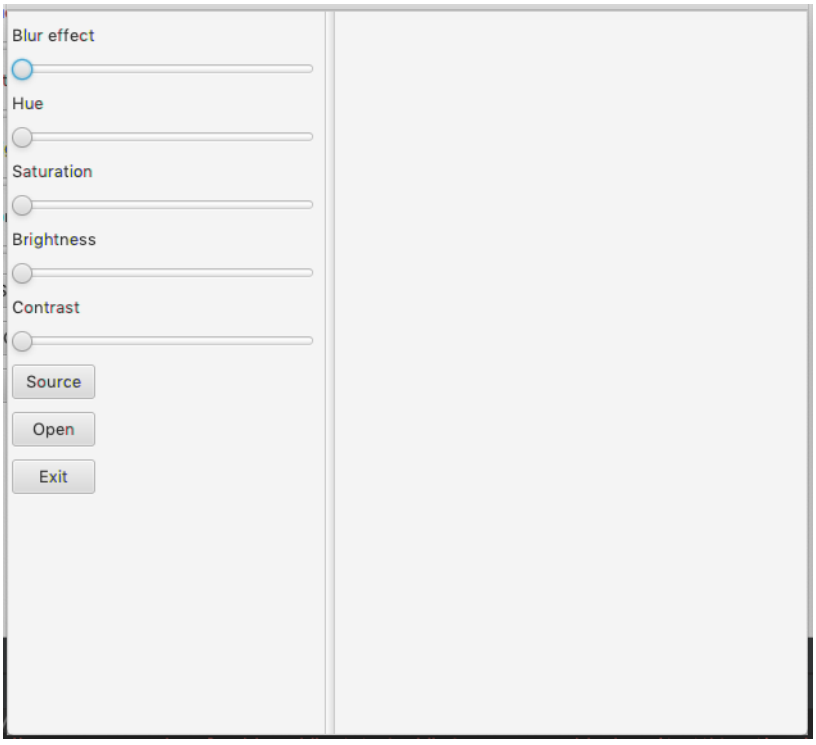

Рисунок 2 – Пример расположения компонентов

2. Открытие изображения из проекта

Подготовьте файл с любым изображением. Разместите его в своем проекте в папке пакета.

Далее в обработчике кнопки “Source” будем загружать его в компонент *ImageView*. Для этого нужно создать объект класса *Image*. В конструкторе необходимо передать путь к изображению. Так как изображение будет лежать в рабочем каталоге программы, то сначала надо получить этот путь. Для этого можно использовать метод *getClass()*. Далее вызываем метод *getResourceAsStream()*, которому нужно указать имя изображения.

```
Image img =
        new Image(getClass().getResourceAsStream("logo.jpg"));
mainImage.setImage(img);
```

Также, если задан путь к файлу, или с ним необходимо провести какие-либо действия, можно использовать класс File.

Например:

```
File file = new File("src/sample/logo.jpg");
```

Далее преобразовать его в строковое представление и передать в конструктор класса Image:

```
Image img = new Image(file.toURI().toString());
```

Если у компонента ImageView не заданы размеры, то он автоматически масштабируется под размер изображения. Если же задать в разметке размеры, то открываемое изображение будет масштабировано под этот размер.

Также можно масштабировать отображаемое изображение в контейнере ImageView в процессе работы в программном коде. Для этог можно использовать методы `setFitHeight()` и `setFitWidth()`.

У экземпляра Stage также можно использовать методы `setMinWidth()` в сочетании с `setMinHeight()`, позволяющие задать минимальные размеры окна.

Для того, чтобы получить доступ к экземпляру Stage, модифицируем код главного класса, добавив ему поле класса Stage. Чтобы к нему можно было получить доступ за пределами кода класса, сделаем его открытым и статическим:

```
private static Stage stage;
```

В методе `start()` класса *Main* присвоим данному полу объект *primaryStage*:

```
stage = primaryStage;
```

Также добавим к этому полю геттер.

3. Открытие изображения с любого источника

FileChooser позволяет пользователю использовать системный диалог, чтобы выбрать один или более файлов. Идентичный компонент это *DirectoryChooser* позволяет пользователю выбирать папку.

У этих диалоговых окон всегда стиль текущей платформы, который независим от JavaFX. На некоторых платформах, где доступ к файлу может быть ограничен (например, на некоторых мобильных или встроенных устройствах), диалоговое окно файла может вернуть вместо файла *null*.

FileChooser может использоваться, чтобы вызвать файл открытые диалоговые окна для того, чтобы выбрать единственный файл (*showOpenDialog*), чтобы выбрать многократные файлы (*showOpenMultipleDialog*) и диалоговые окна сохранения файла (*showSaveDialog*). Конфигурацией выведенного на экран диалогового окна управляют значения *FileChooser*. Эта конфигурация включает заголовок диалогового окна, начальный каталог, фильтры расширений для перечисленных файлов. Возвращаемое значение определяет выбранный файл или равняется *null* если диалоговое окно было отменено.

Для начала создадим экземпляр класса *FileChooser* и установим заголовок будущего окна:

```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Open Image File");
```

Далее создадим фильтры для файлов. Для этого используется подкласс *ExtensionFilter*. В конструкторе задается имя фильтра, а также список файловых масок:

Например:

```
FileChooser.ExtensionFilter filter1 =
     new FileChooser.ExtensionFilter("All image files", "*.png",
"*.jpg", "*.gif");
FileChooser.ExtensionFilter filter2 =
     new FileChooser.ExtensionFilter("JPEG files", "*.jpg");
FileChooser.ExtensionFilter filter3 =
     new    FileChooser.ExtensionFilter("PNG    image    Files",
"*.png");
FileChooser.ExtensionFilter filter4 =
     new    FileChooser.ExtensionFilter("GIF    image    Files",
"*.gif");
```

Созданные фильтры нужно добавить созданному объекту методом *addAll*():

```
fileChooser.getExtensionFilters()
               .addAll(filter1, filter2, filter3, filter4);
```

Теперь можно вызвать его на экран методом *showOpenDialog*(). Данный метод требует указания в качестве параметра объекта класса Stage основной формы приложения. Так как в нашем приложении stage объявлен внутри класса Main, то к нему нет доступа.

Теперь можно вызвать окно диалога:

```
File file =
       fileChooser.showOpenDialog(MainApplication.getStage());
```

После того, как пользователь выберет файл, данный метод вернет объект класса *File*. Для начала, необходимо проверить, что он не пустой. Далее преобразовать его к формату унифицированного указателя ресурсов и передать полученный результат в конструктор класса Image:

```
if (file != null) {
      Image img = new Image(file.toURI().toString());
      mainImage.setImage(img);
}
```

Так как изображение может иметь различный размер, то размер компонента будет автоматически подогнан под размер изображения. При этом, надо изменить размер основного окна, чтобы компонент полностью был виден.

Это можно сделать методами setMinWidth и setMinHeight. Дополнительно можно добавить размер с учетом границ вокруг изображения, и дополнительных элементов управления снизу.

```
Main.stage.setMinWidth(img.getWidth()+90);
Main.stage.setMinHeight(img.getHeight()+160);
```

4. Создание и применение эффекта размытия

Эффект (Effect) это любое действие создающее воздействие на графическое изображение (Graphic image), чтобы создать другое графическое изображение. Один из эффектов, которое вы часто встречаете это эффект размытия (Motion Blur), эффект тени (Drop Shadow), и т.д.

JavaFX множество разных эффектов, все они находятся в одном пакете (package) *javafx.scene.effect*:

Blend, Bloom, BoxBlur, ColorAdjust, ColorInput, DropShadow, GaussianBlur, Glow, ImageInput, InnerShadow, Lighting, MotionBlur, PerspectiveTransform, Reflection, SepiaTone, Shadow.

Рассмотрим работы с классом *BoxBlur* на примере.

Создаем объект сразу в обработчике кнопки и настраиваем его параметры. Настройка параметров размытия и количества проходов:

```
boxBlur.setWidth();
boxBlur.setHeight();
boxBlur.setIterations().
```

Количество итераций влияют на силу размытия. Данную цифру возьмем с компонента *Slider*.

Пример полученного кода:

```
public void onBlur(MouseEvent mouseEvent) {
        boxBlur.setWidth(10);
        boxBlur.setHeight(10);
```

```
        boxBlur.setIterations((int)sliderBlur.getValue());
        mainImage.setEffect(boxBlur);
    }
```

5. Настройки изображения

Так же можно применить к изображению различные эффекты изменения цвета (Color adjust efffect). Чтобы изменить цвет на изображении можно изменить следующие свойства (property):

- hue
- saturation
- brightness
- contrast

Все свойства имеют тип double и изменяются в диапазоне -1..1.

Для применения подобных преобразований к изображениям используется классы ColorAdjust с установленными значениями.

```
colorAdjust.setContrast(double value);
colorAdjust.setHue(double value);
colorAdjust.setBrightness(double value);
colorAdjust.setSaturation(double value);
```

Применить преобразования к изображению можно установив его методом setEffect.

Пример изменения контрастности изображения:

```
ColorAdjust colorAdjust = new ColorAdjust();
public void onContrast(MouseEvent mouseEvent) {
   colorAdjust.setSaturation(sliderSatiration.getValue());
   mainImage.setEffect(colorAdjust);
}
```

Чтобы отменить все преобразования достаточно установить эффект в null.

```
imgView.setEffect(null);
```

Задание.

1. Подготовить приложение из рассмотренного в работе примера.
2. Реализовать логику обработки событий всех компонентов Slilder.
3. Добавить кнопку для сброса всех значений и эффектов.
4. Использовать стили CSS для оформления приложения.