Урок 4

Продвинутые вопросы создания графического интерфейса

Создание собственных элементов управления. Работа с графикой. Обработка событий.

Создание формы

Для создания простого окна достаточно создать класс с наследованием от JFrame, и создать объект этого класса в методе main(), как показано ниже:

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        setVisible(true);
    }
}
public class MainClass {
    public static void main(String[] args) {
        new MyWindow();
    }
}
```

В конструкторе сразу же задаются параметры окна: **setTitle()** — устанавливает заголовок; **setDefaultCloseOperation(WindowConstants.*EXIT_ON_CLOSE*)** — сообщает системе о необходимости завершить работу программы при закрытии окна; **setBounds()** — задаёт координаты и размер формы в пикселях; **setVisible(true)** — показывает форму на экране, желательно вызывать этот метод после всех настроек, иначе при запуске некоторые элементы могут быть отображены некорректно.

Рисунок 1 — Пустое окно Swing

Попробуем добавить на форму несколько управляющих элементов, например, 5 кнопок JButton. Как правило, в библиотеке Swing названия классов, отвечающих за элементы графического интерфейса, начинаются с буквы J.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        JButton[] jbs = new JButton[5];
        for (int i = 0; i < 5; i++) {
            jbs[i] = new JButton("#" + i);
        }
        setLayout(new BorderLayout());    // выбор компоновщика элементов
        add(jbs[0], BorderLayout.EAST);   // добавление кнопки на форму
        add(jbs[1], BorderLayout.WEST);
        add(jbs[2], BorderLayout.SOUTH);
        add(jbs[3], BorderLayout.NORTH);
        add(jbs[4], BorderLayout.CENTER);
        setVisible(true);
    }
}
```

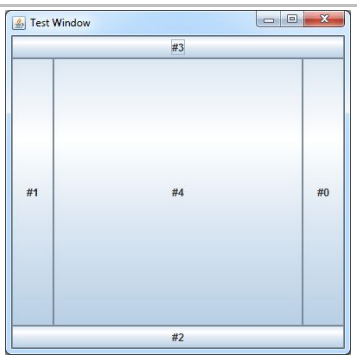

Рисунок 2 — Форма с несколькими кнопками

За расстановку элементов на форме отвечают компоновщики элементов, в данном случае мы использовали BorderLayout. После создания кнопок их необходимо добавить/расположить на форме, для этого используется метод add элемент_интерфейса, местонахождение.

Наиболее используемые компоновщики элементов

BorderLayout — располагает элементы «по сторонам света» (запад, восток, север, юг и центр). Элемент, имеющий расположение CENTER, занимает бОльшую часть окна, то есть при растяжении формы сторонние элементы не будут менять размер, а центральный будет растягиваться, чтобы занять всю имеющуюся площадь.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        JButton button = new JButton("Button 1 (PAGE_START)");
        add(button, BorderLayout.PAGE_START);
        button = new JButton("Button 2 (CENTER)");
        button.setPreferredSize(new Dimension(200, 100));
        add(button, BorderLayout.CENTER);
        button = new JButton("Button 3 (LINE_START)");
        add(button, BorderLayout.LINE_START);
        button = new JButton("Long-Named Button 4 (PAGE_END)");
        add(button, BorderLayout.PAGE_END);
        button = new JButton("5 (LINE_END)");
        add(button, BorderLayout.LINE_END);
        setVisible(true);
    }
}
```

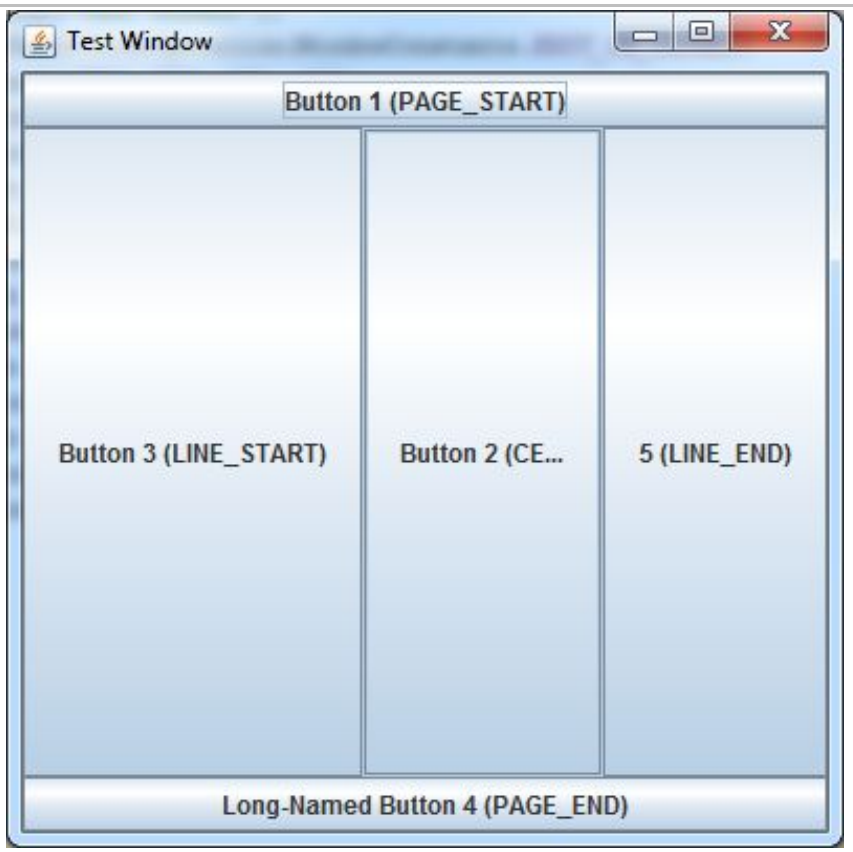

Рисунок 3 — BorderLayout

BoxLayout — располагает элементы в строку или столбец, в зависимости от используемой константы: BoxLayout.Y_AXIS для расположения элементов в столбец, BoxLayout.X_AXIS — в строку.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500,500,500,300);
```

```
        setTitle("BoxLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS)); // одну из
строк надо закомментировать
        setLayout(new BoxLayout(getContentPane(), BoxLayout.X_AXIS)); // одну из
строк надо закомментировать
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            jbs[i].setAlignmentX(CENTER_ALIGNMENT);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

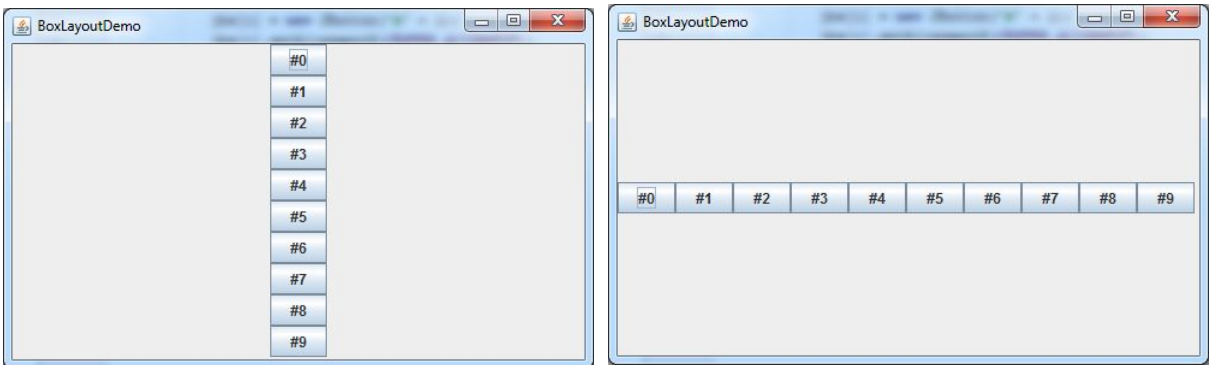

Рисунок 4 — BoxLayout

FlowLayout — располагает элементы в одну строку, когда ширины строки становится недостаточно, переносит новые элементы на следующую.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500, 500, 400, 300);
        setTitle("FlowLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new FlowLayout());
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

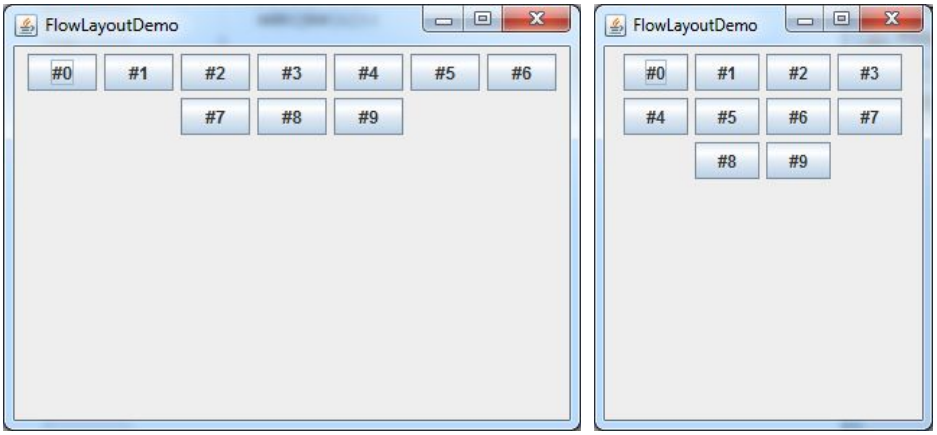

Рисунок 5 — FlowLayout

GridLayout — элементы управления выравниваются по таблице заданного размера. Все строки имеют одинаковую высоту, все столбцы — длину. Элемент управления может занимать только одну ячейку таблицы.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500, 500, 400, 300);
        setTitle("GridLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new GridLayout(4, 3));
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

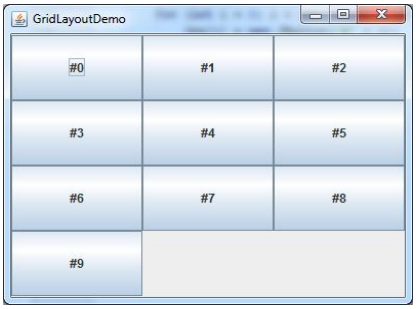

Рисунок 6 — GridLayout

Отключение компоновщика элементов — возможен вариант ручной расстановки элементов путём указания их абсолютных координат и размеров с помощью метода setBounds(), для чего необходимо выключить компоновщик элементов через setLayout(null).

Пример использования базовых элементов интерфейса:

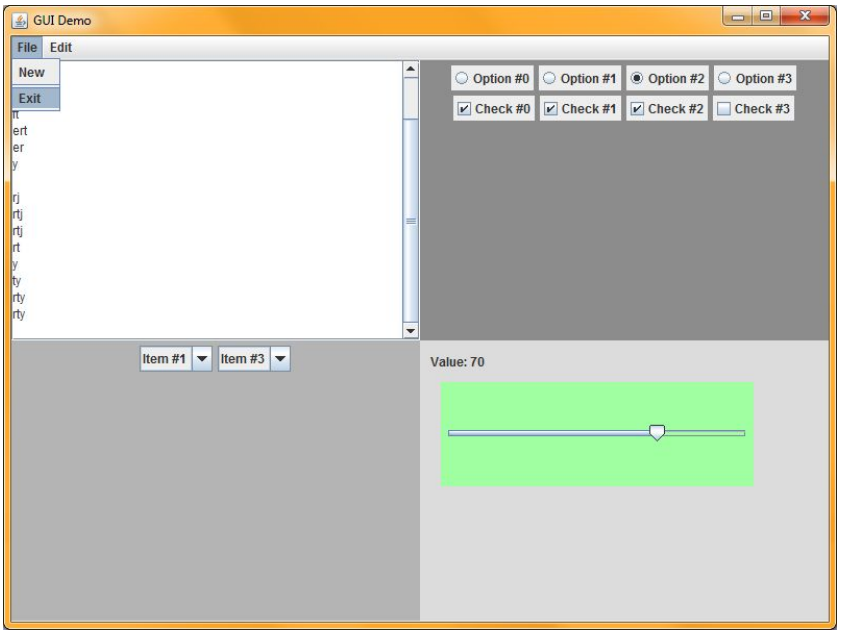

Рисунок 7 — Использование различных элементов интерфейса

В первом блоке кода настраиваем параметры окна и создаём четыре панели для размещения элементов, при этом задаём им отличный друг от друга цвет. Располагаем эти панели на форме в виде таблицы (2, 2), с помощью GridLayout.

```
setBounds(500, 200, 800, 600);
setTitle("GUI Demo");
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
setLayout(new GridLayout(2, 2));
JPanel[] jp = new JPanel[4];
for (int i = 0; i < 4; i++) {
    jp[i] = new JPanel();
    add(jp[i]);
    jp[i].setBackground(new Color(100 + i * 40, 100 + i * 40, 100 + i * 40));
}
```

На первой панели расположено многострочное текстовое поле, которое находится внутри элемента JScrollPane, что позволяет пролистывать контент этого поля.

```
jp[0].setLayout(new BorderLayout());
JTextArea jta = new JTextArea();
JScrollPane jsp = new JScrollPane(jta);
jp[0].add(jsp);
```

Во второй панели содержится 2 типа элементов: JCheckBox и JRadioButton. JCheckBox предназначен для вкл/выкл каких-либо опций (флажков), при этом одновременно может быть включено несколько JCheckBox. JRadioButton предоставляет выбор только одного пункта из набора, т.е. может быть выбрана только одна опция. Для корректной работы связанных RadioButton их необходимо заносить в ButtonGroup.

```
jp[1].setLayout(new FlowLayout());
JRadioButton[] jrb = new JRadioButton[4];
ButtonGroup bgr = new ButtonGroup();
for (int i = 0; i < jrb.length; i++) {
    jrb[i] = new JRadioButton("Option #" + i);
    bgr.add(jrb[i]);
    jp[1].add(jrb[i]);
}
JCheckBox[] jcb = new JCheckBox[4];
for (int i = 0; i < jcb.length; i++) {
    jcb[i] = new JCheckBox("Check #" + i);
    jp[1].add(jcb[i]);
}
```

На третьей панели расположена пара элементов типа JComboBox, которые представляют собой выпадающие списки. ActionListener для JComboBox проверяет событие выбора пользователем одного из пунктов.

```
jp[2].setLayout(new FlowLayout());
String[] comboStr = {"Item #1", "Item #2", "Item #3", "Item #4"};
JComboBox<String> jcombo1 = new JComboBox<String>(comboStr);
JComboBox<String> jcombo2 = new JComboBox<String>(comboStr);
jp[2].add(jcombo1);
jp[2].add(jcombo2);
jcombo1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println(jcombo1.getSelectedItem().toString());
    }
});
```

Четвёртая панель представляет собой пример расстановки элементов с использованием абсолютных координат. На ней расположено обычное нередактируемое текстовое поле, которое показывает значение, выбранное на JSlider.

```
jp[3].setLayout(null);
JSlider js = new JSlider();
JLabel jlab = new JLabel("Value: 50");
js.setMaximum(100);
js.setMinimum(0);
js.setValue(50);
jp[3].add(jlab);
jp[3].add(js);
js.addChangeListener(new ChangeListener() {
    @Override
    public void stateChanged(ChangeEvent e) {
        jlab.setText("Value: " + js.getValue());
    }
});
jlab.setBounds(10, 10, 100, 20);
js.setBounds(20, 40, 300, 100);
js.setBackground(new Color(160, 255, 160));
```

Последним пунктом идёт создание верхнего меню окна. Для этого создаются элементы JMenuBar -> JMenu -> JMenuItem. Как видно из кода ниже, для обработки нажатия на один из подпунктов меню достаточно «повесить» на него ActionListener.

```
JMenuBar mainMenu = new JMenuBar();
JMenu mFile = new JMenu("File");
JMenu mEdit = new JMenu("Edit");
JMenuItem miFileNew = new JMenuItem("New");
JMenuItem miFileExit = new JMenuItem("Exit");
setJMenuBar(mainMenu);
mainMenu.add(mFile);
mainMenu.add(mEdit);
mFile.add(miFileNew);
mFile.addSeparator(); // разделительная линия в меню
mFile.add(miFileExit);

miFileExit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("BYE");
    }
});
```

Обработка событий

Обработка событий является неотъемлемой частью разработки прикладных программ с графическим пользовательским интерфейсом (ГПИ). Любая прикладная программа с ГПИ выполняется под управлением событий, большинство которых направлено на взаимодействие с пользователем. Существует несколько типов событий, включая генерируемые мышью, клавиатурой, различными элементами интерфейса. Рассмотрим некоторые варианты обработки событий.

Обработка кликов по кнопке

```
JButton button = new JButton("Button");
add(button);
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed...");
    }
});
```

Для отслеживания кликов по кнопке необходимо добавить ActionListener, как показано выше. Как только произойдет событие нажатия этой кнопки, выполнится метод actionPerformed().

Обработка нажатия кнопки Enter в текстовом поле

```
JTextField field = new JTextField();
add(field);
field.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Your message: " + field.getText());
    }
});
```

При работе с текстовым полем (TextField) ActionListener отлавливает нажатие кнопки Enter, конечно, только в случае, если поле находится в фокусе. Поэтому нет необходимости отслеживать именно нажатие кнопки Enter, например, через addKeyListener(…) с указанием кода этой клавиши.

Отслеживание кликов мыши по панели и определение координат клика

```
JPanel panel = new JPanel();
add(panel);
panel.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        System.out.println("MousePos: " + e.getX() + " " + e.getY());
    }
});
```

В приведённом примере отслеживание отжатия кнопки мыши над объектом типа JPanel. В объект e типа MouseEvent попадает вся информацию о произошедшем событии, в том числе, координатах курсора в системе координат панели, кнопке, которая была нажата (левая или правая), количестве кликов (одинарный, двойной) и т.д.

Отслеживание момента закрытия окна

```
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("Bye");
    }
});
```

Чтобы выполнить какое-либо действие при закрытии окна, необходимо к объекту типа JFrame добавить WindowListener. В примере выше ссылка на объект отсутствует, так как метод прописан в конструкторе класса, наследуемого от JFrame.

Работа с JavaFX

Для разработки графического интерфейса, помимо Swing в Java существует возможность использовать библиотеку JavaFX, для работы с которой крайне желательно использовать JDK 8 (в более поздних версиях JavaFX вынесли из JDK в отдельный модуль). Создать JavaFX проект можно прямо из IntelliJ IDEA.

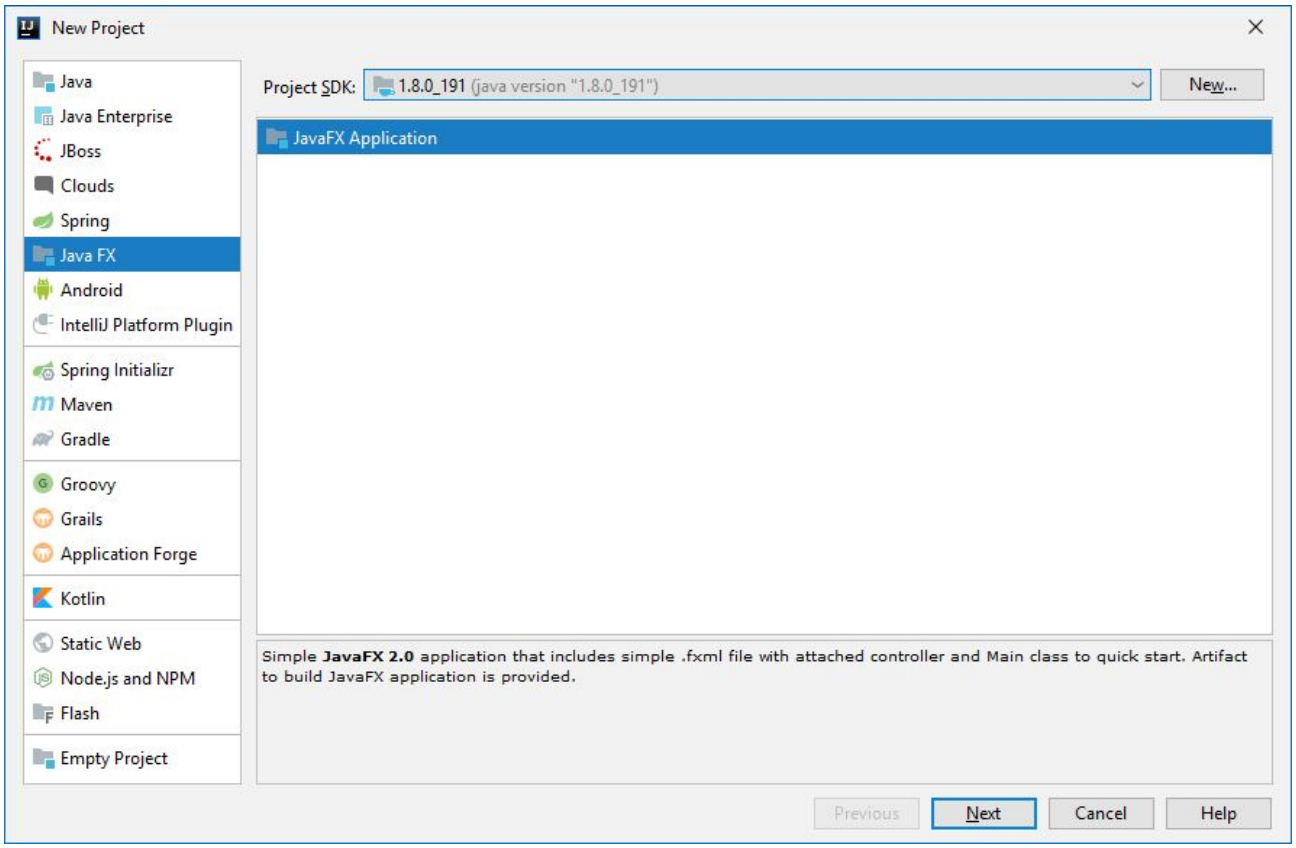

Рисунок 8 - Подготовка проекта

После настройки проекта, получим следующую структуру:

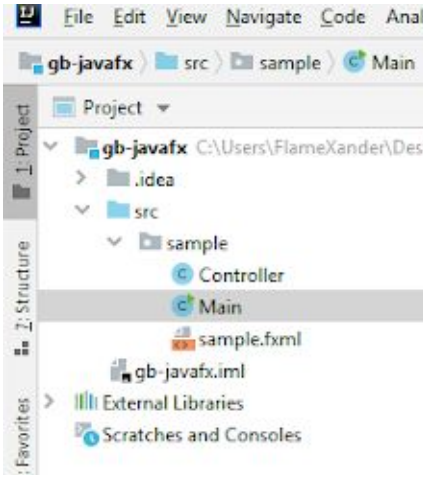

Рисунок 9 - Структура проекта

В проект входит 2 класса и один fxml файл. Разберем по-порядку что для чего нужно. Класс Main служит для запуска JavaFX приложения, его код представлен ниже:

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception{
```

```
        Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root, 300, 275));
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Метод main() запускает приложение, и с помощью метода launch() запускается сам графический интерфейс. В методе start() производится загрузка сцены из файла sample.fxml. Сцена представляет собой набор элементов управления, которые будут добавлены на форму. Если в Swing создание элементов управления прописывалось в коде, то в JavaFX описание интерфейса вынесено в отдельный файл формата fxml. Stage primaryStage представляет собой ссылку на основное окно приложения (аналог класса JFrame из Swing). Для окна настраивается его заголовок через setTitle() и устанавливается сцена и ее размер с помощью метода setScene(). После выполнения подготовительных работ, форма отображается на экране методом show().

Что из себя представляет fxml файл?

```
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.TextArea?>

<VBox fx:controller="sample.Controller" xmlns:fx="http://javafx.com/fxml"
alignment="center">
   <TextArea VBox.vgrow="ALWAYS" />
   <Button text="Button #1" />
</VBox>
```

В fxml файл “зашито” описание элементов управления на сцене. Первым элементом в данном случае является VBox - это панель-компоновщик, располагающая внутренние элементы по вертикали друг за другом. В эту панель добавлено два элемента: TextArea (многострочное текстовое поле) и Button (кнопка). Каждый из элементов может быть настроен с помощью набора атрибутов.

Таблица 1 - Примеры атрибутов для кнопки (Button)

Имя атрибута	**Значение**
String text	Надпись на кнопке
float minWidth, maxWidth	Минимально и максимально допустимые значения длины
boolean disable	Отключение
boolean visible	Видимость
float opacity (0.0f - 1.0f)	Прозрачность

Таблица 2 - Примеры атрибутов для многострочного текстового поля (TextArea)

Имя атрибута	Значение
String text	Содержимое текстового поля
float minWidth, maxWidth	Минимально и максимально допустимые значения длины
boolean disable	Отключение
boolean visible	Видимость
float opacity (0.0f - 1.0f)	Прозрачность
boolean editable	Вкл/откл возможности изменения текста

Если элемент управления расположен внутри панели-компоновщика, то он может быть настроен с помощью атрибутов, привязанных к этому типу панели. В данном примере элементы расположены внутри VBox, и следовательно могут быть настроены под этот компоновщик. TextArea имеет атрибут VBox.vgrow=”ALWAYS”, что означает что текстовое поле должно быть максимально возможно растянуто по вертикали.

У корневого элемента VBox прописан атрибут fx:controller, указывающий на класс-контроллер, который будет управлять сценой, описанной в данном fxml файле.

Часто встречающиеся компоновщики элементов:

- **BorderPane** - аналог BorderLayout из Swing;
- **HBox** - располагает элементы по горизонтали друг за другом;
- **VBox** - располагает элементы по вертикали друг за другом;
- **GridPane** - располагает элементы по сетке (аналог GridLayout из Swing);
- **FlowPane** - располагает элементы друг за другом по горизонтали или вертикали, если элементу не хватает места на текущей линии, то производится перенос на следующую линию. (при горизонтальном расположении перенос на следующую строку, при вертикальном - в следующий столбец);
- **AnchorPane** - позволяет привязывать элементы к сторонам панели.

Часто используемые элементы управления:

- **Button** - кнопка;
- **TextArea** - многострочное текстовое поле;
- **TextField** - однострочное текстовое поле;
- **ListView** - список элементов;
- **TableView** - таблица;
- **Label** - надпись;

Как добавить действие на нажатие кнопки

Чтобы повесить ActionListener на элемент управления в JavaFX добавить к элементу управления добавить атрибут onAction.

```
<?import javafx.geometry.Insets?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
```

```
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.TextArea?>

<VBox fx:controller="sample.Controller" xmlns:fx="http://javafx.com/fxml"
alignment="center">
   <TextArea VBox.vgrow="ALWAYS" />
   <Button text="Button #1" onAction="#btnOneClickAction"/>
</VBox>
```

Такая за запись означает, что при нажатии на кнопку должен сработать метод btnOneClickAction(), который прописан в контроллере sample.Controller. До тех пор, пока вы не реализуете этот метод, в fxml файле он будет гореть красным цветом.

Давайте добавим метод btnOneClickAction().

```
package sample;

import javafx.event.ActionEvent;

public class Controller {
   public void btnOneClickAction(ActionEvent actionEvent) {
       System.out.println(1);
   }
}
```

Теперь при нажатии на кнопку, в консоль будет выводиться строка “1”. Аргумент ActionEvent actionEvent хранит в себе информацию о произошедшем событии.

```
public void btnOneClickAction(ActionEvent actionEvent) {
  Button thisButton = (Button)actionEvent.getSource(); // Вот так можно получить
ссылку на кнопку, которая была нажата (например, если вы одно и то же действие
повесили на десяток кнопок)
}
```

С помощью атрибута onAction в fxml файле можно добавлять действия для всех элементов управления. Но нужно учесть, что для **Button (кнопки)** onAction() это событие нажатия на эту кнопку; для **TextField (однострочного текстового поля)** - событие нажатия кнопки Enter в этом текстовом поле; для **Slider (слайдера)** - событие изменения его значения, перемещения бегунка.

Как из контроллера обратиться к элементу управления

Для начала, в fxml файле, элементу присваивается специальный идентификатор fx:id.

```
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.TextArea?>

<VBox fx:controller="sample.Controller" xmlns:fx="http://javafx.com/fxml"
alignment="center">
   <TextArea fx:id="mainTextArea" VBox.vgrow="ALWAYS" />
   <Button text="Button #1" onAction="#btnOneClickAction"/>
</VBox>
```

С помощью этого идентификатора мы можем связать контроллер и fxml документ. Чтобы из контроллера сослаться на элемент управления, используйте аннотацию @FXML.

```
package sample;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;

public class Controller {
    @FXML
    TextArea mainTextArea;

    public void btnOneClickAction(ActionEvent actionEvent) {
        mainTextArea.appendText("1\n");
    }
}
```

В контроллере объявляется элемент управления соответствующего типа (TextArea), над его объявлением ставится аннотация @FXML, а имя поля должно обязательно совпадать с fx:id в fxml файле. Все готово, теперь через mainTextArea вы можете работать с текстовым полем на форме. В примере выше, нажатие на кнопку будет приводить к печати строки “1” в TextArea.
Заметка: если вдруг вы сделали все что требуется, а строка класс TextArea в строке `TextArea mainTextArea;` горит красным, то обязательно проверьте import, скорее всего вы импортировали не javafx.scene.control.TextArea, а java.awt.TextArea.

Практическое задание

1. Создать окно для клиентской части чата: большое текстовое поле для отображения переписки в центре окна. Однострочное текстовое поле для ввода сообщений и кнопка для отсылки сообщений на нижней панели. Сообщение должно отсылаться либо по нажатию кнопки на форме, либо по нажатию кнопки Enter. При «отсылке» сообщение перекидывается из нижнего поля в центральное.

Дополнительные материалы

1. Вебинар GeekBrains: Разработка простого файлового менеджера на Java + JavaFX: https://geekbrains.ru/events/1804
2. Вебинар GeekBrains: Разработка простого файлового менеджера на Java + JavaFX (чуть более продвинутый вариант): https://geekbrains.ru/events/1792

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Г. Шилдт. Java 8. Полное руководство // 9-е изд.: Пер. с англ. — М.: Вильям с , 2015. — 1 376 с.