

Лабораторная работа №6

РАЗРАБОТКА JAVA-ПРИЛОЖЕНИЙ (ЧАСТЬ 1)

Цель работы:

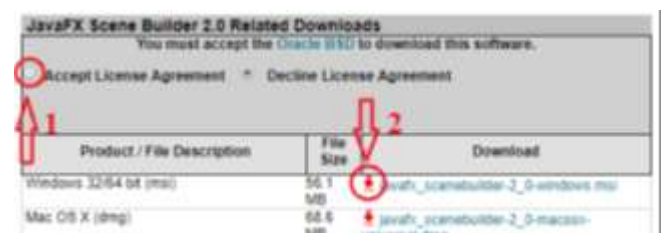
Создание приложения на JavaFX с использованием редактора интерфейса Scene Builder.

Указания к работе:

- Создание и запуск проекта JavaFX;
- Использование приложения Scene Builder для проектирования пользовательского интерфейса;
- Простая структуризация приложения с использованием шаблона MVC.

Подключим все необходимое для дальнейшей работы:

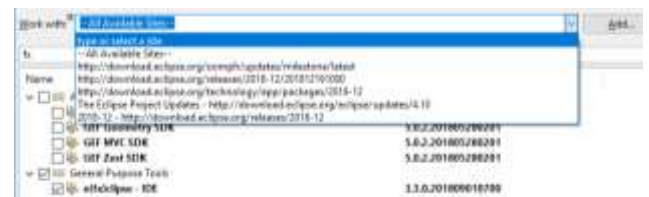
1. Установить **Scene Builder 2.0** по ссылке
<https://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html>



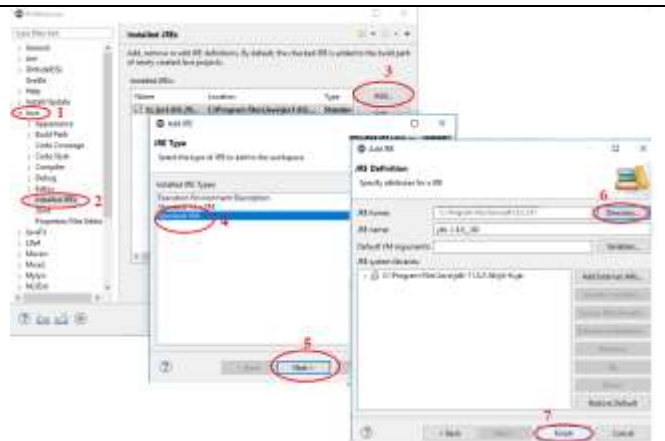
2. Установить **JDK 1.8** по ссылке
<https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html>

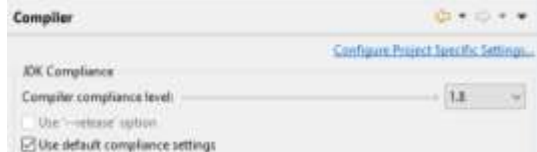
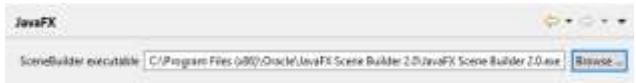


3. Откройте среду разработки Eclipse и перейдите к пункту **Help | Install new software**. В открывшемся окне выберите из списка **--All Available Sites--**. Найдите среди предложенных обновлений **General Purpose Tools** и выберите **e(fx)clipse – IDE** нажмите **Finish**.



4. Перейдите в меню **Window | Preferences** далее в правом меню **Java | Installed JREs**. Нажмите **Add...**, выберите **Standard VM** и укажите путь к установленной **JDK 1.8**. Уберите другие добавленные и JDK будет использоваться по умолчанию. Нажмите **Apply**.



5. Перейдите к пункту Java Compiler . Установите значение настройки Compiler compliance level в 1.8 . Нажмите Apply .	
6. Перейдите к пункту JavaFX и укажите путь к исполняемому файлу приложения Scene Builder .	

1. Создание нового проекта JavaFX

В приложение Eclipse (с уже установленным e(fx)clipse) в меню выберите пункт **File | New | Other...**, и затем выберите **JavaFX Project**. Укажите имя проекта (наше будет называться **StudentGroupApp**). Поставьте галочку на **Use default JRE** и нажмите **Finish**. Если Eclipse автоматически создало какие-то начальные файлы и пакеты, то удалите их.

2. Создание структуры пакетов

Создадим шаблон проектирования Модель-Представление-Контроллер (MVC). Опираясь на этот шаблон мы разобьём код нашего приложения на три части и создадим для каждой из них свой пакет (правый клик на папке **src, New... | Package** и создаем следующие пакеты):

studentgroup - содержит большинство классов-контроллеров (Controller)

studentgroup.model - содержит классы Моделей (Model);

studentgroup.view - содержит Представления (View). Также содержит некоторые классы-контроллеры, которые непосредственно связаны с конкретными представлениями (view-controllers).

3. Создание файла разметки FXML

Есть два пути создания пользовательского интерфейса: либо использовать файл разметки FXML, либо программировать всё на Java. Для визуального редактирования наших XML-файлов будем использовать Scene Builder.

Кликните на пакете **studentgroup.view** правой кнопкой мышки и создайте новый документ **FXML** с названием **PersonOverview**.

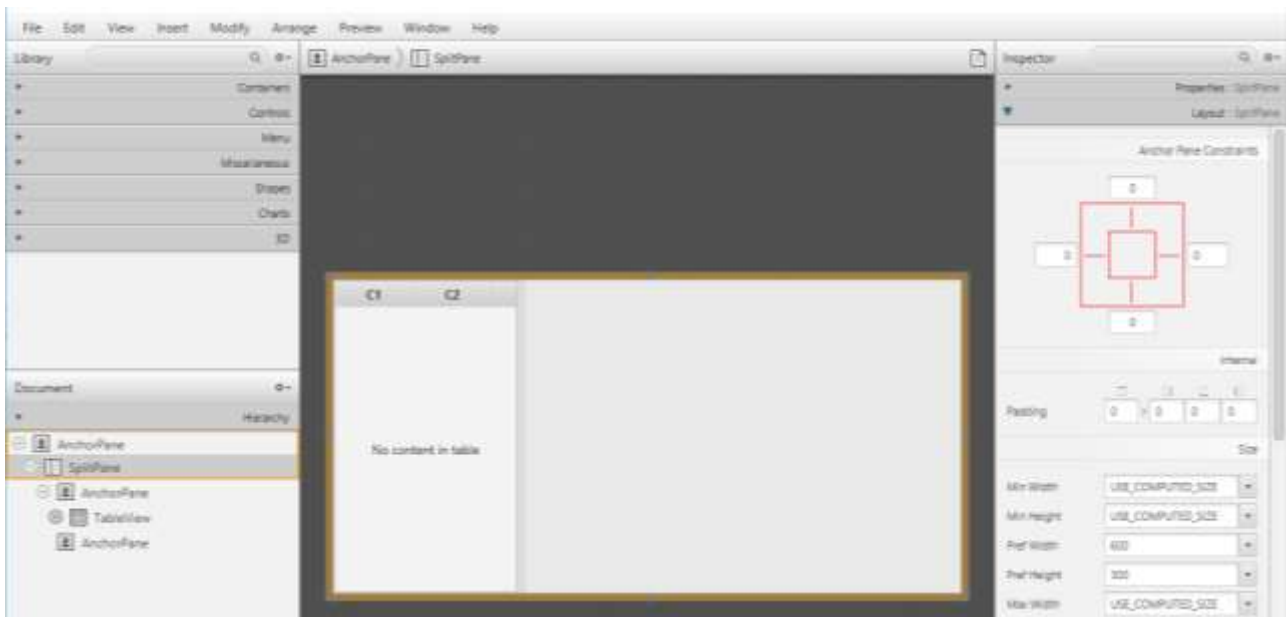
4. Проектировка визуального интерфейса в Scene Builder

Откройте только что созданный fxml-документ в приложении Scene Builder - клик правой кнопкой мышки по файлу **PersonOverview.fxml**, **Open with SceneBuilder**. На вкладке **Hierarchy** (внизу слев) должен находиться единственный компонент **AnchorPane**. (Если **Scene Builder** не запустился, то открываем пункт меню **Window | Preferences | JavaFX** и настраиваем верный путь к исполняемому файлу установленного приложения **Scene Builder**).

- На вкладке **Hierarchy** выберите компонент **AnchorPane**, и справа, на вкладке **Layout** установите значения характеристикам **Pref Width** и **Pref Height** - 600 и 300 соответственно.

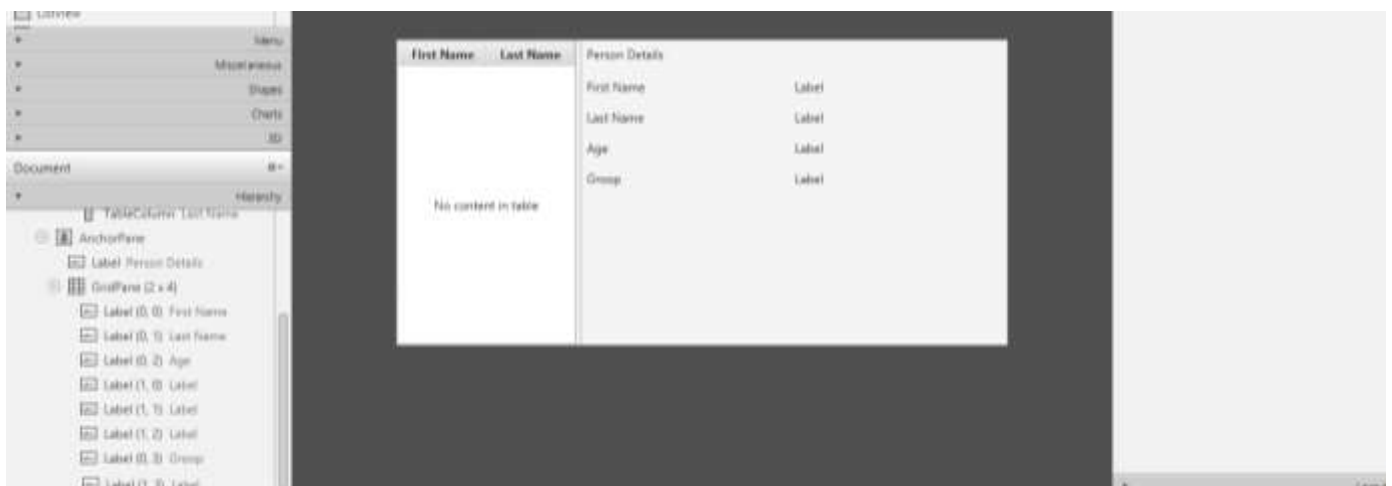
- Добавьте на **AnchorPane** элемент **SplitPane (horizontal)** из вкладки **Containers** (вверху слева). На вкладке **Hierarchy** кликните по нему правой кнопкой мыши и выберите **Fit to Parent**.

- Добавьте в левую часть компонента **SplitPane** со вкладки **Controls** компонент **TableView**. Выделите его целиком во вкладке **Hierarchy** (а не отдельный столбец) и проставьте отступы от краёв = **0** так, как показано на рисунке. Внутри компонента



AnchorPane всегда можно проставить отступы от четырёх границ рамки.

- В таблице измените заголовки колонок (вкладка **Properties** компонента **TableColumn**) на «**First Name**» и «**Last Name**».
- Выберите компонент **TableView** и во вкладке **Properties** измените значение **Column Resize Policy** на **constrained-resize** (теперь колонки таблицы всегда будут занимать всё доступное пространство).
- В правую часть компонента **SplitPane** перетащите компонент **Label** и измените его текст на «**Person Details**». Используя привязки к границам (вкладка **Layout**) скорректируйте его положение («**5**» сверху и слева, остальное без изменений).
- На правую панель **SplitPane** добавьте компонент **GridPane** и так же настройте привязки к границам («**5**» справа и слева, «**30**» с верха, низ без изменений).
- Приведите своё окно в соответствие с тем, что показано на рисунке, добавляя компоненты **Label** внутрь ячеек компонента **GridPane** (чтобы добавить новый ряд в компонент **GridPane**, выберите существующий номер ряда (он окрасится жёлтым), кликните правой кнопкой мышки на номере ряда и выберите пункт «**Add Row Above**» или «**Add Row Below**»).

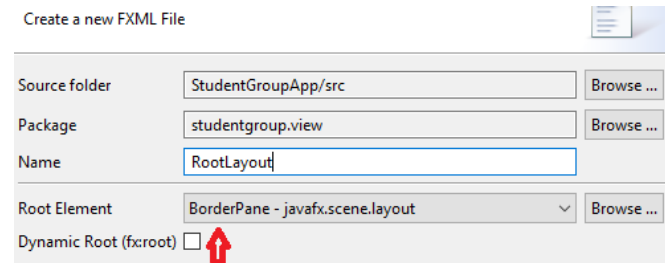


Добавьте три компонента **Button** в правую часть. Выделите их всех вместе (Shift + клик), кликните по ним правой кнопкой мышки и выберите пункт **Wrap In | HBox**. Это действие их сгруппирует. Вы можете задать расстояние (**Spacing**) между компонентами во вкладке **Properties** компонента **HBox**. Также установите привязки к границам (правой и нижней).

Используйте пункт меню **Preview**, чтобы протестировать созданное окно и его реакцию на изменение размеров.

5. Создание основного приложения.

- В пакете **studentgroup.view** создайте другой fxml-документ, и назовите его **RootLayout.fxml**. На этот раз в качестве корневого (**Root Element**) элемента выберите **BorderPane**. Это обертка для первого fxml-документа, будет содержать меню.
- Откройте только что созданный файл в **Scene Builder**. Установите значение **Pref Width** и **Pref Height** - 600 и 400 соответственно.
- В верхний слот компонента **BorderPane** добавьте компонент **MenuBar** (функциональность панели будет описано позже).



6. Основной класс приложения JavaFX.

- Создадим класс Java, который будет запускать наше приложение с **RootLayout.fxml** и добавлять в его центральную область **PersonOverview.fxml**. Для этого ПКМ по проекту **StudentGroupApp**, **New | Other... | JavaFX Main Class**. поместите его в пакет **studentgroup** (пакет является родительским для view и model).

Созданный класс **MainApp.java** расширяет класс **Application** и содержит два метода. Это базовая структура, которая необходима для запуска приложения JavaFX. Нам интересен метод **start(Stage primaryStage)**. Он автоматически вызывается при вызове метода **launch(...)** из метода **main**. **Stage** является основным контейнером, который, как правило, представляет собой обрамлённое окно со стандартными кнопками: закрыть, свернуть, развернуть. Внутри **Stage** добавляется сцена **Scene**, которая может быть заменена другой **Scene**. Внутри **Scene** добавляются стандартные компоненты типа **AnchorPane**, **TextBox** и другие.

- Откройте класс **MainApp.java** и замените его содержимое на это:

```
package studentgroup;

import java.io.IOException;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MainApp extends Application {

    private Stage primaryStage;
    private BorderPane rootLayout;

    @Override
```

```

public void start(Stage primaryStage) {
    this.primaryStage = primaryStage;
    this.primaryStage.setTitle("StudentGroupApp");

    initRootLayout();

    showPersonOverview();
}

/**
 * Инициализирует корневой макет.
 */
public void initRootLayout() {
    try {
        // Загружаем корневой макет из fxml файла.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/RootLayout.fxml"));
        rootLayout = (BorderPane) loader.load();

        // Отображаем сцену, содержащую корневой макет.
        Scene scene = new Scene(rootLayout);
        primaryStage.setScene(scene);
        primaryStage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Показывает в корневом макете сведения о студентах.
 */
public void showPersonOverview() {
    try {
        // Загружаем сведения о студентах.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/PersonOverview.fxml"));
        AnchorPane personOverview = (AnchorPane) loader.load();

        // Помещаем сведения о студентах в центр корневого макета.
        rootLayout.setCenter(personOverview);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Возвращает главную сцену.
 * @return
 */
public Stage getPrimaryStage() {
    return primaryStage;
}

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

```

Если JavaFX не может найти указанный fxml-файл, то вы получите следующее сообщение об ошибке:

java.lang.IllegalStateException: Location is not set.

Для решения этой проблемы внимательно проверьте правильность указания пути к файлам fxml и правильность написания его названия.

- **Создание класса-модели.**

Класс-модель необходим для хранения информации о студентах. Добавьте класс **Person.java** в пакет **studentgroup.model**. В нём будет несколько переменных для хранения информации об имени, группе и возрасте.

```
package studentgroup.model;

import java.time.LocalDate;

import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.StringProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.beans.property.SimpleStringProperty;

/**
 * Класс-модель для студента (Person).
 */
public class Person {

    private final StringProperty firstName;
    private final StringProperty lastName;
    private final IntegerProperty groupCode;
    private final ObjectProperty<LocalDate> birthday;

    /**
     * Конструктор по умолчанию.
     */
    public Person() {
        this(null, null);
    }

    /**
     * Конструктор с некоторыми начальными данными.
     *
     * @param firstName
     * @param lastName
     */
    public Person(String firstName, String lastName) {
        this.firstName = new SimpleStringProperty(firstName);
        this.lastName = new SimpleStringProperty(lastName);

        // Какие-то фиктивные начальные данные для удобства тестирования.
        this.groupCode = new SimpleIntegerProperty(1747);
        this.birthday = new SimpleObjectProperty<LocalDate>(LocalDate.of(2000, 2, 21));
    }

    public String getFirstName() {
        return firstName.get();
    }

    public void setFirstName(String firstName) {
        this.firstName.set(firstName);
    }

    public StringProperty firstNameProperty() {
        return firstName;
    }

    public String getLastName() {
        return lastName.get();
    }

    public void setLastName(String lastName) {
        this.lastName.set(lastName);
    }
}
```



```

public StringProperty lastNameProperty() {
    return lastName;
}

public int getGroupCode() {
    return groupCode.get();
}

public void setGroupCode(int postalCode) {
    this.groupCode.set(postalCode);
}

public IntegerProperty groupCodeProperty() {
    return groupCode;
}

public LocalDate getBirthday() {
    return birthday.get();
}

public void setBirthday(LocalDate birthday) {
    this.birthday.set(birthday);
}

public ObjectProperty<LocalDate> birthdayProperty() {
    return birthday;
}
}

```

В JavaFX для всех полей класса-модели предпочтительно использовать **Properties**; Класс **LocalDate**, для переменной **birthday**, это часть Date and Time API. Основные данные приложения - это группа экземпляров класса **Person**. Создадим в классе **MainApp.java** список объектов класса **Person**. Все остальные классы-контроллеры позже получат доступ к этому центральному списку внутри этого класса.

- Список **ObservableList**

Классы-представления в JavaFX, необходимо информировать при любых изменениях в списке студентов. Это важно, потому что, не будь этого, мы бы не смогли синхронизировать представление данных с самими данными. Для этой цели в JavaFX были введены некоторые новые классы коллекций. Из этих классов нам понадобится класс **ObservableList**. Для создания экземпляра данного класса добавьте приведённый код в начало **MainApp.java**. Мы так же добавим в код конструктор, который будет создавать некоторые демонстрационные данные и метод-геттер с публичным модификатором доступа:

```

// ... ДОБАВИТЬ К СПИСКУ ИМПОРТА СЛЕДУЮЩИЕ 3 СТРОКИ...
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import studentgroup.model.Person;

// ... ВСТАВИТЬ ПОСЛЕ (private BorderPane) ...
/**
 * Данные, в виде наблюдаемого списка адресатов.
 */
private ObservableList<Person> personData = FXCollections.observableArrayList();

/**
 * Конструктор
 */
public MainApp() {
    // В качестве образца добавляем некоторые данные
    personData.add(new Person("Петр", "Пяточкин"));
    personData.add(new Person("Иван", "Зайцев"));
}

```

```

        personData.add(new Person("Екатерина", "Васильченко"));
        personData.add(new Person("Ольга", "Жук"));
        personData.add(new Person("Людмила", "Алексеева"));
        personData.add(new Person("Данил", "Кац"));
        personData.add(new Person("Евгений", "Васнецов"));
        personData.add(new Person("Дмитрий", "Жуликов"));
        personData.add(new Person("Мрат", "Алибов"));
        personData.add(new Person("Martin", "Mueller"));
    }

    /**
     * Возвращает данные в виде наблюдаемого списка студентов.
     * @return
     */
    public ObservableList<Person> getPersonData() {
        return personData;
    }

    // ... ДАЛЕЕ ОСТАЛЬНАЯ ЧАСТЬ КЛАССА MainApp.java...

```

- **Класс PersonOverviewController**

Создайте новый класс **PersonOverviewController.java**. внутри пакета **view**. Этот класс-контроллер для представления **PersonOverview.fxml** применяем для отображения в таблице некоторых данных (разместить в том же пакете, где находится файл разметки **PersonOverview.fxml**, иначе Scene Builder не сможет его найти).

Определяем переменные для доступа к таблице и меткам представления. Эти переменные и некоторые методы имеют специальную аннотацию **@FXML**. Она необходима для того, чтобы fxml-файл имел доступ к приватным полям и методам. Цель настроить fxml-файл так, что при его загрузке приложение автоматически заполняло эти переменные данными.

```

package studentgroup.view;

import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import studentgroup.MainApp;
import studentgroup.model.Person;

public class PersonOverviewController {
    @FXML
    private TableView<Person> personTable;
    @FXML
    private TableColumn<Person, String> firstNameColumn;
    @FXML
    private TableColumn<Person, String> lastNameColumn;

    @FXML
    private Label firstNameLabel;
    @FXML
    private Label lastNameLabel;
    @FXML
    private Label groupCodeLabel;
    @FXML
    private Label birthdayLabel;

    // Ссылка на главное приложение.
    private MainApp mainApp;

    /**
     * Конструктор.
     * Конструктор вызывается раньше метода initialize().
     */
}

```



```

public PersonOverviewController() {
}

/**
 * Инициализация класса-контроллера. Этот метод вызывается автоматически
 * после того, как fxml-файл будет загружен.
 */
@FXML
private void initialize() {
    // Инициализация таблицы студентов с двумя столбцами.
    firstNameColumn.setCellValueFactory(cellData ->
cellData.getValue().firstNameProperty());
    lastNameColumn.setCellValueFactory(cellData ->
cellData.getValue().lastNameProperty());
}

/**
 * Вызывается главным приложением, которое даёт на себя ссылку.
 *
 * @param mainApp
 */
public void setMainApp(MainApp mainApp) {
    this.mainApp = mainApp;

    // Добавление в таблицу данных из наблюдаемого списка
    personTable.setItems(mainApp.getPersonData());
}
}

```

Все поля и методы, к которым fxml-файлу потребуется доступ, должны быть отмечены аннотацией `@FXML`. Несмотря на то, что это требование предъявляется только для полей и методов с модификатором `private`, лучше оставить их закрытыми и пометить аннотацией, чем делать публичными!

После загрузки fxml-файла автоматически вызывается метод `initialize()`. На этот момент все FXML-поля должны быть инициализированы;

Метод `setCellValueFactory(...)` определяет, какое поле внутри класса **Person** будут использоваться для конкретного столбца в таблице. Стрелка `->` означает, что мы использовали лямбда-выражение из Java 8. (Есть вариант сделать то же самое через `PropertyValueFactory`, но этот способ нарушает безопасность типов).

*В нашем примере для столбцов таблицы мы использовали только значения **StringProperty**. Если нам понадобится использовать **IntegerProperty** или **DoubleProperty**, то `setCellValueFactory(...)` должен иметь дополнительный метод `asObject()`:*

```

myIntegerColumn.setCellValueFactory(cellData ->
cellData.getValue().myIntegerProperty().asObject());

```

- Соединение класса **MainApp** с классом **PersonOverviewController**

Метод `setMainApp(...)` должен быть вызван из класса **MainApp**. Это даст нашему контроллеру доступ к экземпляру **MainApp**, к коллекции записей **personList** внутри него и к другим элементам класса. Добавьте в метод `showPersonOverview()` после строки `rootLayout.setCenter(personOverview);` две дополнительные строки:

```

// Даём контроллеру доступ к главному приложению.
PersonOverviewController controller = loader.getController();
controller.setMainApp(this);

```

А к списку импорта добавляем: `import studentgroup.view.PersonOverviewController;`

- Привязка класса-контроллера к fxml-файлу

Укажем **PersonOverview.fxml** какой контроллер он должен использовать, и зададим соответствие между элементами представления и полями внутри класса-контроллера. Для этого:

- откройте файл **PersonOverview.fxml** в приложении **Scene Builder**;

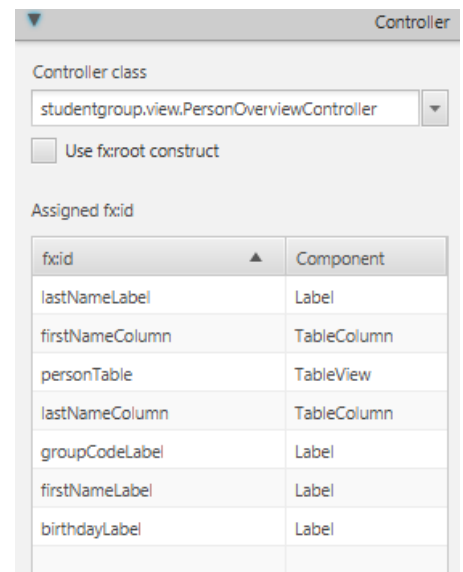
- откройте вкладку **Controller** слева на панели **Document** и выберите класс **PersonOverviewController** в качестве класса-контроллера;

- выберите компонент **TableView** на вкладке **Hierarchy**, перейдите на вкладку **Code** и в поле **fx:id** установите значение **personTable**;

- сделайте то же самое для колонок таблицы и установите значения свойства **fx:id** **firstNameColumn** и **lastNameColumn** соответственно;

- для каждой **Label** во второй колонке компонента **GridPane** также установите соответствующие значения **fx:id**.

- сохраните файл **PersonOverview.fxml**, вернитесь в среду разработки **Eclipse** и обновите весь проект **StudentGroupApp** (F5). Это необходимо для того, чтобы приложение **Eclipse** «увидело» те изменения, которые мы сделали в приложении **Scene Builder**.



7. Обработка реакции выбора студента из списка.

- Создайте метод **showPersonDetails(Person person)** для заполнения меток данными указанного студента (Person). Используя метод **setText(...)** метод присваивает меткам значения из переданного в параметре объекта Person. Если в качестве параметра передаётся **null**, то весь текст в метках будет очищен.

Добавим новый метод в конец уже существующего класса **PersonOverviewController**:

```
/**
 * Заполняет все текстовые поля, отображая подробности о студенте.
 * Если указанный студент = null, то все текстовые поля очищаются.
 *
 * @param person — студент типа Person или null
 */
private void showPersonDetails(Person person) {
    if (person != null) {
        // Заполняем метки информацией из объекта person.
        firstNameLabel.setText(person.getFirstName());
        lastNameLabel.setText(person.getLastName());
        groupCodeLabel.setText(Integer.toString(person.getGroupCode()));

        // TODO: Нам нужен способ для перевода дня рождения в тип String!
        // birthdayLabel.setText(...);
    } else {
        // Если Person = null, то убираем весь текст.
        firstNameLabel.setText("");
        lastNameLabel.setText("");
        groupCodeLabel.setText("");
        birthdayLabel.setText("");
    }
}
```

- Преобразование дня рождения в строку

Создадим новый пакет **studentgroup.util**. Мы не можем присвоить текстовой метке значение поля **birthday**, так как тип его значения **LocalDate** а не **String**, необходимо отформатировать дату. Создадим в новом пакете класс **DateUtil**.

```

package studentgroup.util;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;

/**
 * Вспомогательные функции для работы с датами.
 *
 * @author Marco Jakob
 */
public class DateUtil {

    /** Шаблон даты, используемый для преобразования. Можно поменять на свой. */
    private static final String DATE_PATTERN = "dd.MM.yyyy";

    /** Форматировщик даты. */
    private static final DateTimeFormatter DATE_FORMATTER =
        DateTimeFormatter.ofPattern(DATE_PATTERN);

    /**
     * Возвращает полученную дату в виде хорошо отформатированной строки.
     * Используется определённый выше {@link DateUtil#DATE_PATTERN}.
     *
     * @param date - дата, которая будет возвращена в виде строки
     * @return отформатированную строку
     */
    public static String format(LocalDate date) {
        if (date == null) {
            return null;
        }
        return DATE_FORMATTER.format(date);
    }

    /**
     * Преобразует строку, которая отформатирована по правилам
     * шаблона {@link DateUtil#DATE_PATTERN} в объект {@link LocalDate}.
     *
     * Возвращает null, если строка не может быть преобразована.
     *
     * @param dateString - дата в виде String
     * @return объект даты или null, если строка не может быть преобразована
     */
    public static LocalDate parse(String dateString) {
        try {
            return DATE_FORMATTER.parse(dateString, LocalDate::from);
        } catch (DateTimeParseException e) {
            return null;
        }
    }

    /**
     * Проверяет, является ли строка корректной датой.
     *
     * @param dateString
     * @return true, если строка является корректной датой
     */
    public static boolean validate(String dateString) {
        // Пытаемся разобрать строку.
        return DateUtil.parse(dateString) != null;
    }
}

```

Формат даты можно поменять, просто изменив константу `DATE_PATTERN`. Все возможные форматы описаны в документации к классу `[DateTimeFormatter]`

Теперь мы можем использовать наш новый класс **DateUtil** в методе **showPersonDetails** класса **PersonOverviewController**. Замените комментарий **TODO** следующей строкой:

```
birthdayLabel.setText(DateUtil.format(person.getBirthday()));
```

А в импорт добавить: `import studentgroup.util.DateUtil;`

- «Прослушивание» изменений.

Для этого в **JavaFX** существует интерфейс **ChangeListener** с единственным методом **changed(...)**. Этот метод имеет три параметра: **observable**, **oldValue** и **newValue**.

Интерфейс **ChangeListener** реализован с помощью лямбда-выражений, изменим методу **initialize()** класса **PersonOverviewController**:

```
@FXML
private void initialize() {
    // Инициализация таблицы адресатов с двумя столбцами.
    firstNameColumn.setCellValueFactory(
        cellData -> cellData.getValue().firstNameProperty());
    lastNameColumn.setCellValueFactory(
        cellData -> cellData.getValue().lastNameProperty());

    // Очистка дополнительной информации о студенте.
    showPersonDetails(null);

    // Слушаем изменения выбора, и при изменении отображаем
    // дополнительную информацию о студенте.
    personTable.getSelectionModel().selectedItemProperty().addListener(
        (observable, oldValue, newValue) -> showPersonDetails(newValue));
}
```

Если мы передаём в параметр метода **showPersonDetails(...)** значение **null**, то все значения меток будут стёрты. В строке **personTable.getSelectionModel...** мы получаем **selectedItemProperty** таблицы и добавляем к нему слушателя. Когда пользователь выбирает запись в таблице, выполняется наше лямбда-выражение. Мы берём только что выбранную запись и передаём её в метод **showPersonDetails(...)**.

Запустите свое приложение и проверьте, отображаются ли данные по выбранному студенту в правой части, когда в таблице выбирается определённый студент.

8. Кнопка Delete

- Добавим в конец класса **PersonOverviewController** метод удаления из списка, а уже потом назначим его обработчиком кнопки **Delete** в **Scene Builder**.

```
/**
 * Вызывается, когда пользователь кликает по кнопке удаления.
 */
@FXML
private void handleDeletePerson() {
    int selectedIndex = personTable.getSelectionModel().getSelectedIndex();
    personTable.getItems().remove(selectedIndex);
}
```

- В приложении **Scene Builder** откройте файл **PersonOverview.fxml**. Выберите кнопку **Delete**, откройте вкладку **Code** и укажите метод **handleDeletePerson** в значение пункта **On Action**.

- Обработка ошибок при удалении.

Если сейчас запустить приложение, и нажать Удалить не выбрав ни одного студента вылетит исключение **ArrayIndexOutOfBoundsException**, потому что не

получится удалить адресата с индексом **-1**. Значение **-1** возвращается методом **getSelectedIndex()**, когда в таблице ничего не выделено.

Добавим диалоговое окно, которое получит пользователь, в случае если нажмет на **Delete** при не выбранном студенте. Изменим метод **handleDeletePerson**:

```
/**
 * Вызывается, когда пользователь кликает по кнопке удаления.
 */
@FXML
private void handleDeletePerson() {
    int selectedIndex = personTable.getSelectionModel().getSelectedIndex();
    if (selectedIndex >= 0) {
        personTable.getItems().remove(selectedIndex);
    } else {
        // Ничего не выбрано.
        Alert alert = new Alert(AlertType.WARNING);
        alert.initOwner(mainApp.getPrimaryStage());
        alert.setTitle("No Selection");
        alert.setHeaderText("No Person Selected");
        alert.setContentText("Please select a person in the table.");

        alert.showAndWait();
    }
}
```

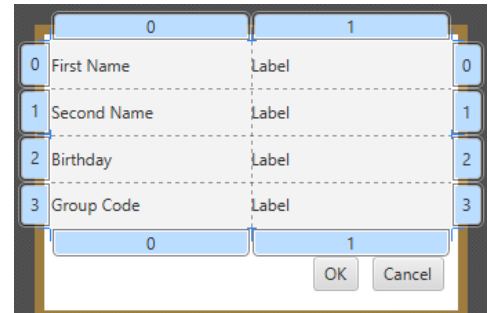
А в импорт добавить:

```
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
```

9. Добавление и изменения списка студентов и их данных

Создаем сцену с формой, содержащую поля для заполнения всей необходимой информации.

- Внутри пакета **view** создайте новый fxml-файл **PersonEditDialog**, в качестве **root Element** ставим **AnchorPane**.
- Используйте компоненты **GridPane**, **Label**, **TextField** и **Button** для создания окна редактирования
- Создание класса контроллера **PersonEditDialogController**



```
package studentgroup.view;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import studentgroup.model.Person;
import studentgroup.util.DateUtil;

/**
 * Окно для изменения информации о студенте.
 */
public class PersonEditDialogController {

    @FXML
    private TextField firstNameField;
    @FXML
    private TextField lastNameField;
    @FXML
    private TextField groupCodeField;
    @FXML
    private TextField birthdayField;
```

```

private Stage dialogStage;
private Person person;
private boolean okClicked = false;

/**
 * Инициализирует класс-контроллер. Этот метод вызывается автоматически
 * после того, как fxml-файл будет загружен.
 */
@FXML
private void initialize() {

}

/**
 * Устанавливает сцену для этого окна.
 *
 * @param dialogStage
 */
public void setDialogStage(Stage dialogStage) {
    this.dialogStage = dialogStage;
}

/**
 * Задаёт адресата, информацию о котором будем менять.
 *
 * @param person
 */
public void setPerson(Person person) {
    this.person = person;

    firstNameField.setText(person.getFirstName());
    lastNameField.setText(person.getLastName());
    groupCodeField.setText(Integer.toString(person.getGroupCode()));
    birthdayField.setText(DateUtil.format(person.getBirthDay()));
    birthdayField.setPromptText("dd.mm.yyyy");
}

/**
 * Returns true, если пользователь кликнул ОК, в другом случае false.
 *
 * @return
 */
public boolean isOkClicked() {
    return okClicked;
}

/**
 * Вызывается, когда пользователь кликнул по кнопке ОК.
 */
@FXML
private void handleOk() {
    if (isInputValid()) {
        person.setFirstName(firstNameField.getText());
        person.setLastName(lastNameField.getText());
        person.setGroupCode(Integer.parseInt(groupCodeField.getText()));
        person.setBirthDay(DateUtil.parse(birthdayField.getText()));

        okClicked = true;
        dialogStage.close();
    }
}

/**
 * Вызывается, когда пользователь кликнул по кнопке Cancel.
 */
@FXML
private void handleCancel() {
    dialogStage.close();
}

```



```

}

/**
 * Проверяет пользовательский ввод в текстовых полях.
 *
 * @return true, если пользовательский ввод корректен
 */
private boolean isValid() {
    String errorMessage = "";

    if (firstNameField.getText() == null || firstNameField.getText().length() == 0) {
        errorMessage += "No valid first name!\n";
    }
    if (lastNameField.getText() == null || lastNameField.getText().length() == 0) {
        errorMessage += "No valid last name!\n";
    }
    if (groupCodeField.getText() == null || groupCodeField.getText().length() == 0) {
        errorMessage += "No valid postal code!\n";
    } else {
        // пытаемся преобразовать номер группы в int.
        try {
            Integer.parseInt(groupCodeField.getText());
        } catch (NumberFormatException e) {
            errorMessage += "No valid group code (must be an integer)!\n";
        }
    }

    if (birthdayField.getText() == null || birthdayField.getText().length() == 0) {
        errorMessage += "No valid birthday!\n";
    } else {
        if (!DateUtil.isValidDate(birthdayField.getText())) {
            errorMessage += "No valid birthday. Use the format dd.mm.yyyy!\n";
        }
    }

    if (errorMessage.length() == 0) {
        return true;
    } else {
        // Показываем сообщение об ошибке.
        Alert alert = new Alert(AlertType.ERROR);
        alert.initOwner(dialogStage);
        alert.setTitle("Invalid Fields");
        alert.setHeaderText("Please correct invalid fields");
        alert.setContentText(errorMessage);

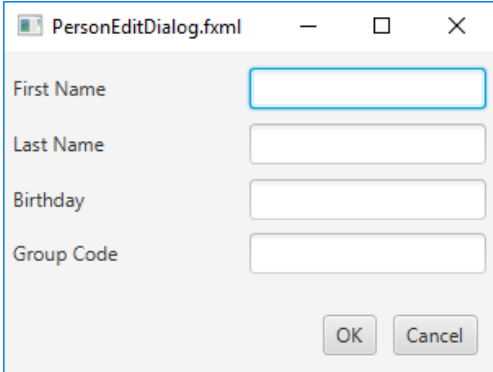
        alert.showAndWait();

        return false;
    }
}
}

```

Для указания студента, данные которого должны быть изменены, метод **setPerson(...)** может быть вызван из другого класса; когда пользователь нажимает на кнопку **OK**, то вызывается метод **handleOK()**. Первым делом данные, введённые пользователем, проверяются в методе **isValid()**. Если проверка прошла успешно, то объект студента заполняется данными, которые ввёл пользователь. Эти изменения будут напрямую применяться к объекту студента, который был передан в качестве аргумента метода **setPerson(...)**!

Логическая переменная **okClicked** служит для определения того, какую из двух кнопок, **OK** или **Cancel** нажал пользователь.



The screenshot shows a JavaFX window titled "PersonEditDialog.fxml". It contains four text input fields labeled "First Name", "Last Name", "Birthday", and "Group Code". At the bottom right, there are two buttons: "OK" and "Cancel". The "First Name" field is currently selected with a blue border.

Привязка класса-контроллера к fxml-файлу.

- Откройте файл **PersonEditDialog.fxml** в Scene Builder;
- С левой стороны во вкладке **Controller** установите наш класс **PersonEditDialogController** в качестве значения параметра **Controller Class**;
- Установите соответствующие значения **fx:id** для всех компонентов **TextField**;
- В значениях параметров **onAction** для кнопок укажите соответствующие методы-обработчики.

• **Вызов диалога редактирования**

Добавьте в класс **MainApp** метод для загрузки и отображения диалога редактирования записей:

```
/**
 * Открывает диалоговое окно для изменения данных указанного студента.
 * Если пользователь кликнул ОК, то изменения сохраняются в предоставленном
 * объекте студента и возвращается значение true.
 *
 * @param person - объект студента, который надо изменить
 * @return true, если пользователь кликнул ОК, в противном случае false.
 */
public boolean showPersonEditDialog(Person person) {
    try {
        // Загружаем fxml-файл и создаём новую сцену
        // для всплывающего диалогового окна.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/PersonEditDialog.fxml"));
        AnchorPane page = (AnchorPane) loader.load();

        // Создаём диалоговое окно Stage.
        Stage dialogStage = new Stage();
        dialogStage.setTitle("Edit Person");
        dialogStage.initModality(Modality.WINDOW_MODAL);
        dialogStage.initOwner(primaryStage);
        Scene scene = new Scene(page);
        dialogStage.setScene(scene);

        // Передаём студента в контроллер.
        PersonEditDialogController controller = loader.getController();
        controller.setDialogStage(dialogStage);
        controller.setPerson(person);

        // Отображаем диалоговое окно и ждём, пока пользователь его не закроет
        dialogStage.showAndWait();

        return controller.isOkClicked();
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}
```

А в импорт добавить:

```
import javafx.stage.Modality;
import studentgroup.view.PersonEditDialogController;
```

- Добавьте следующие методы в класс **PersonOverviewController**. Когда пользователь будет нажимать на кнопки **New...** или **Edit...**, эти методы будут обращаться к методу **showPersonEditDialog(...)** в классе **MainApp**.

```
/**
 * Вызывается, когда пользователь кликает по кнопке New...
 * Открывает диалоговое окно с дополнительной информацией нового студента.
 */
@FXML
private void handleNewPerson() {
```

```

        Person tempPerson = new Person();
        boolean okClicked = mainApp.showPersonEditDialog(tempPerson);
        if (okClicked) {
            mainApp.getPersonData().add(tempPerson);
        }
    }

    /**
     * Вызывается, когда пользователь кликает по кнопка Edit...
     * Открывает диалоговое окно для изменения выбранного студента.
     */
    @FXML
    private void handleEditPerson() {
        Person selectedPerson = personTable.getSelectionModel().getSelectedItem();
        if (selectedPerson != null) {
            boolean okClicked = mainApp.showPersonEditDialog(selectedPerson);
            if (okClicked) {
                showPersonDetails(selectedPerson);
            }

        } else {
            // Ничего не выбрано.
            Alert alert = new Alert(AlertType.WARNING);
            alert.initOwner(mainApp.getPrimaryStage());
            alert.setTitle("No Selection");
            alert.setHeaderText("No Person Selected");
            alert.setContentText("Please select a person in the table.");

            alert.showAndWait();
        }
    }
}

```

- В приложении **Scene Builder** откройте представление **PersonOverview.fxml** и для кнопок **New...** и **Edit...** задайте соответствующие методы-обработчики в параметре **On Action**.

Сохраняйте все и запускайте проект. Должно получиться работающее приложение базы данных студентов. Оно позволяет добавлять, изменять и удалять студентов. Это приложение так же осуществляет проверку всего, что вводит пользователь.

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

1. По аналогии создать свою базу, добавить три поля с типом данных **String**, **Integer** и **Double**.
2. Реализовать возможность добавления, редактирование и удаление всех полей для каждого студента.
3. Реализовать появление диалогового окна, которое ожидает повторное подтверждения от пользователя в случае удаления из списка («Вы уверены, что хотите удалить запись?»).