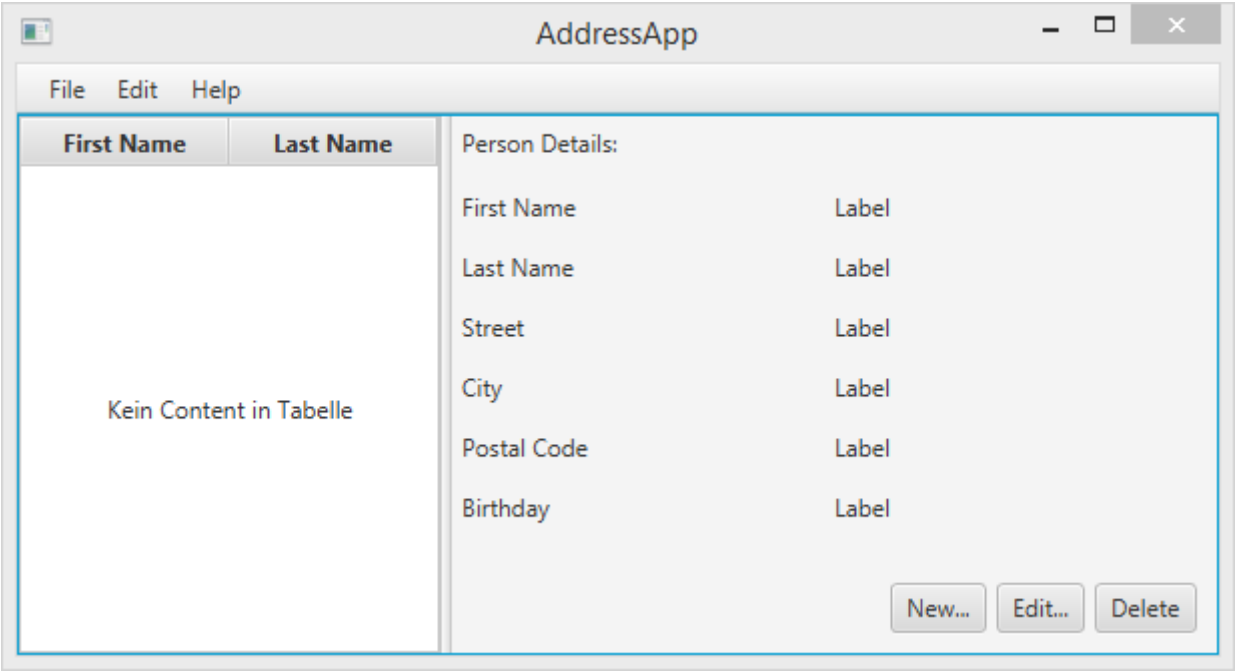
JavaFX 8 - Часть 1: Scene Builder



Часть 1: Содержание

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

Предварительные требования

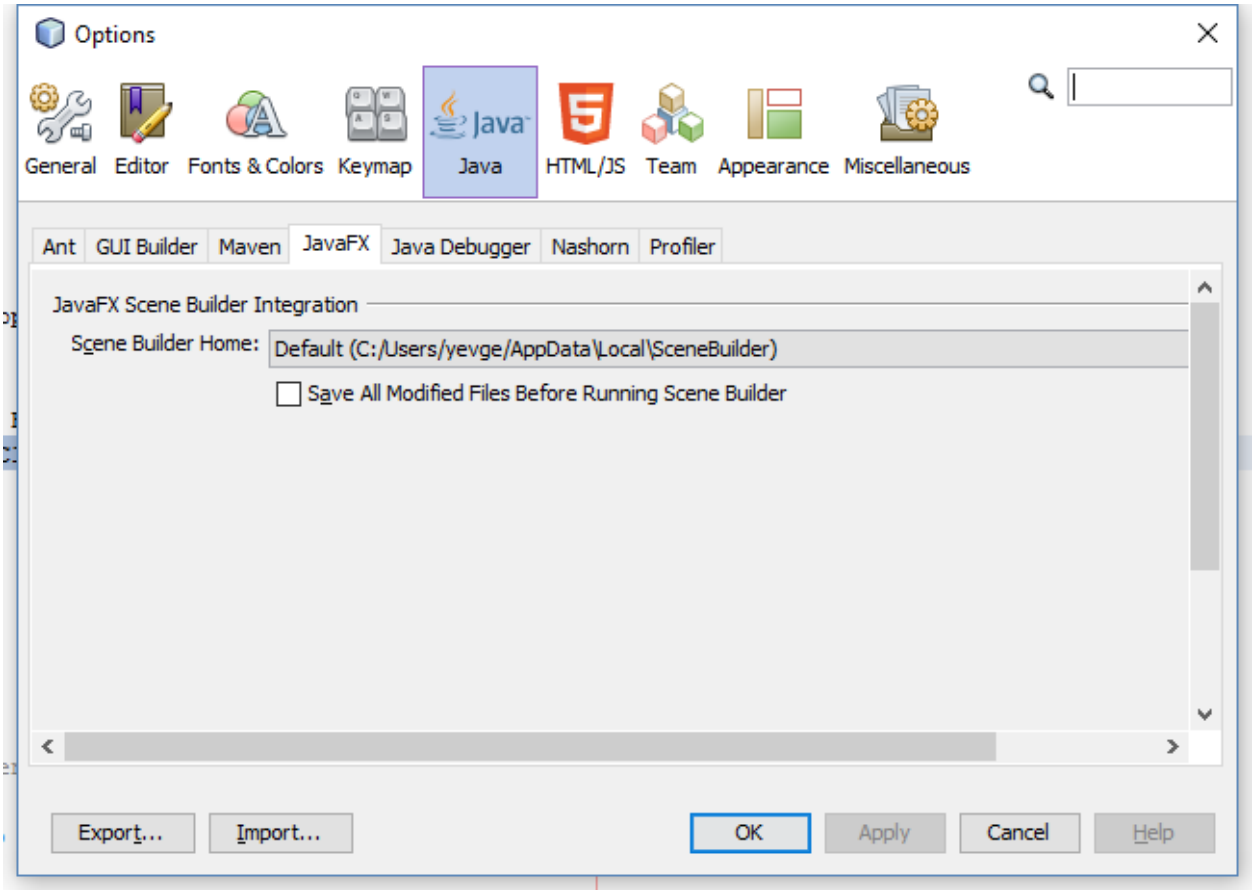
- Последняя Java JDK 8 (включающая в себя **JavaFX 8**);
- Среда разработки
- Приложение Scene Builder версии 8.0 или новее. Сейчас оно предоставляется Gluon, потому как Oracle теперь распространяет его только в виде исходного кода.

Настройка среды разработки Netbeans

Нам нужно указать среде разработки NetBeans *(Tools-Options)* задать путь к приложению Scene Builder:

1. Откройте настройки среды разработки NetBeans и перейдите к пункту *Java*.

2. Перейдите к пункту *JavaFX* и укажите путь к **исполняемому файлу** приложения Scene Builder.



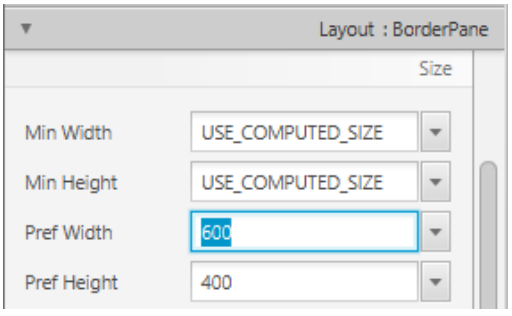
Создание нового проекта JavaFX

В приложении NetBeans в меню выберите пункт *File* | *New Project* | *JavaFX...*, и затем выберите *JavaFX FXML Application и нажмите Next*.
Укажите имя проекта (наше будет называться AddressApp) и нажмите *Finish*.

Проектировка визуального интерфейса в Scene Builder

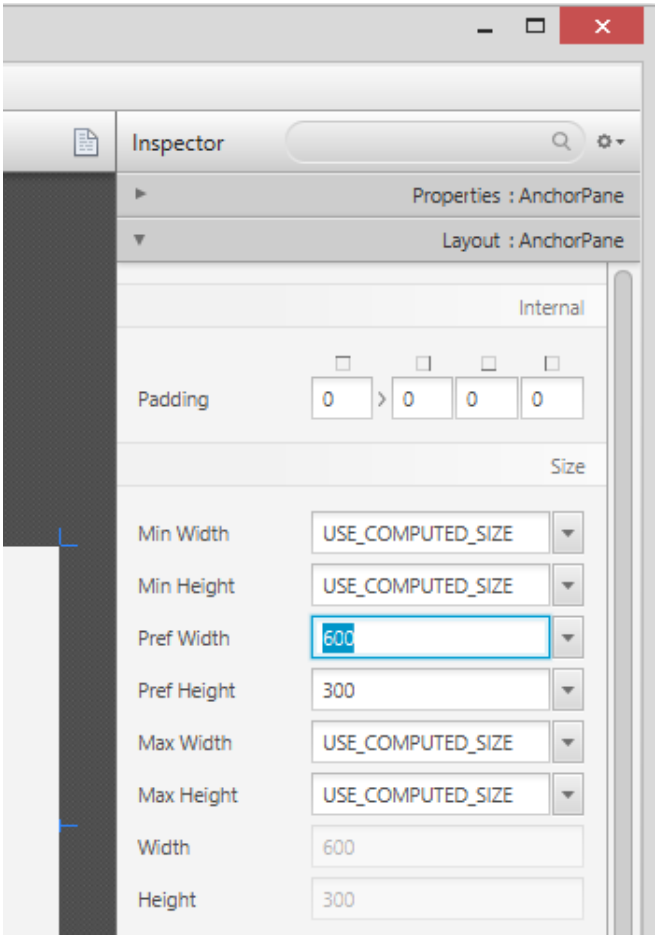
Откройте только что созданный fxml-документ в приложении Scene Builder - клик правой кнопкой мышки по файлу `FXMLDocument.fxml`, *Open with SceneBuilder*. На вкладке *Hierarchy* должен находиться единственный компонент *AnchorPane* с вложенными в него компонентами *Button* и *Label*, который вы должны удалить (правая кнопка мыши - Delete).

1. Из вкладки Containers добавьте в пустую зону компонент BorderPane
2. Установите предпочитаемое значение ширины и высоты компонента: 600 и 400 соответственно.

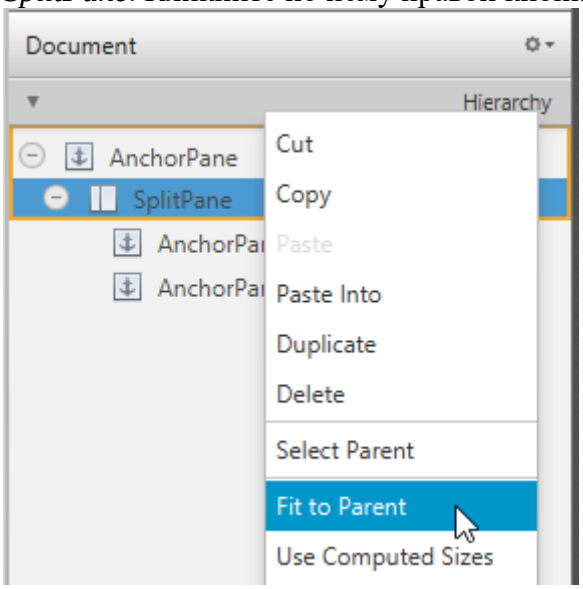


3. В верхний слот компонента *BorderPane* добавьте компонент *MenuBar* из закладки *Controls*. Функциональность меню мы будем реализовывать в последующих уроках.

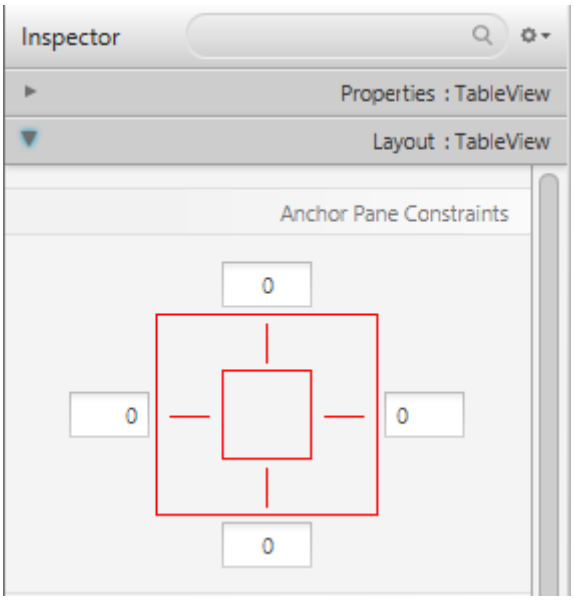
4. В центральный слот компонента *BorderPane* добавьте компонент *AnchorPane* из закладки Containers
5. На вкладке *Hierarchy* выберите компонент *AnchorPane*, и справа, на вкладке *Layout* установите значение характеристикам *Pref Width* и *Pref Height* - 600 и 300 соответственно.



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

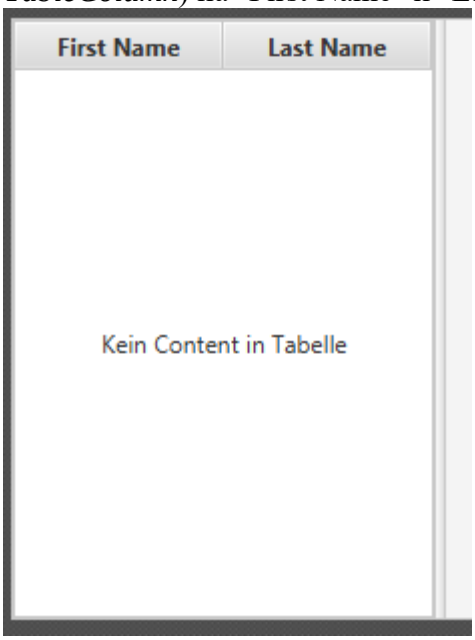


7. На вкладке *Hierarchy* в компонент *SplitPane* добавьте два компонента *AnchorPane*
8. Теперь, в левую часть компонента *SplitPane* со вкладки *Controls* перетащите компонент *TableView*. Выделите его целиком (а не отдельный столбец) и проставьте отступы от краёв так, как показано на рисунке. Внутри компонента *AnchorPane* всегда можно проставить отступы от четырёх границ рамки

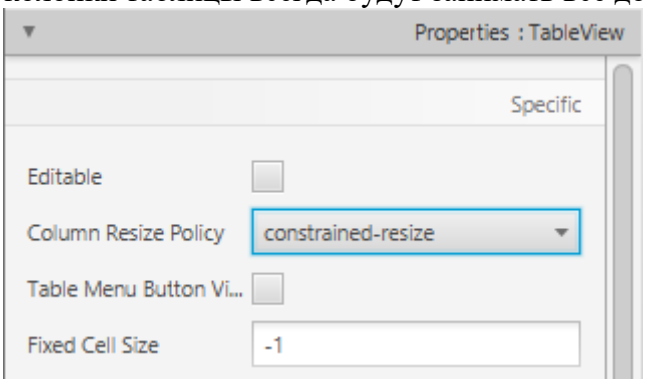


9. Чтобы увидеть, правильно ли отображается созданное окно, выполните пункт меню *Preview | Show Preview in Window*. Попробуйте поменять размер окна. Добавленная таблица должна изменятся вместе с окном, так как она прикреплена к границам окна.

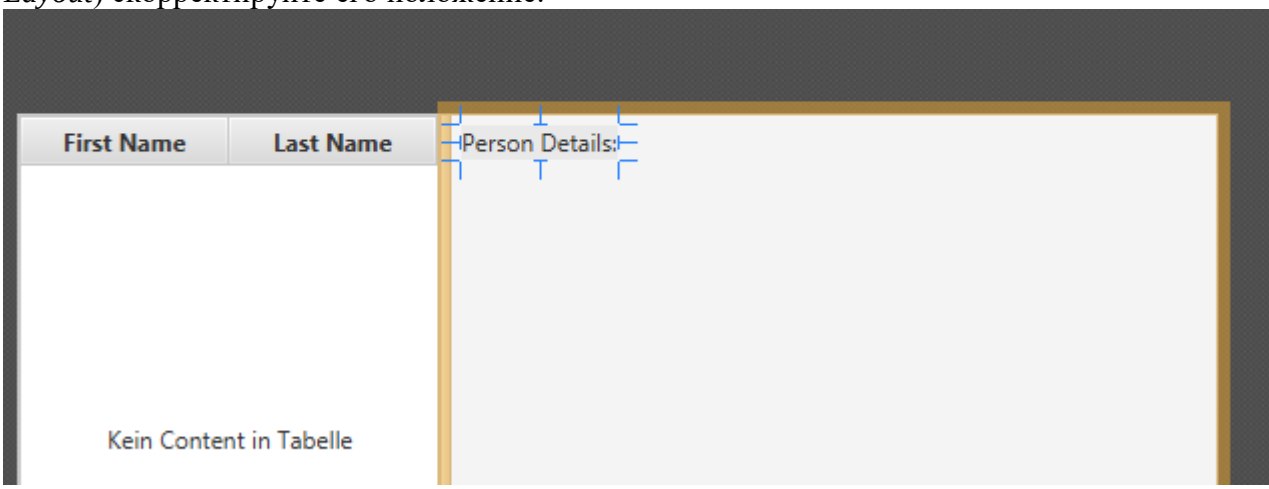
10. В таблице измените заголовки колонок (вкладка *Properties* компонента *TableColumn*) на "First Name" и "Last Name".



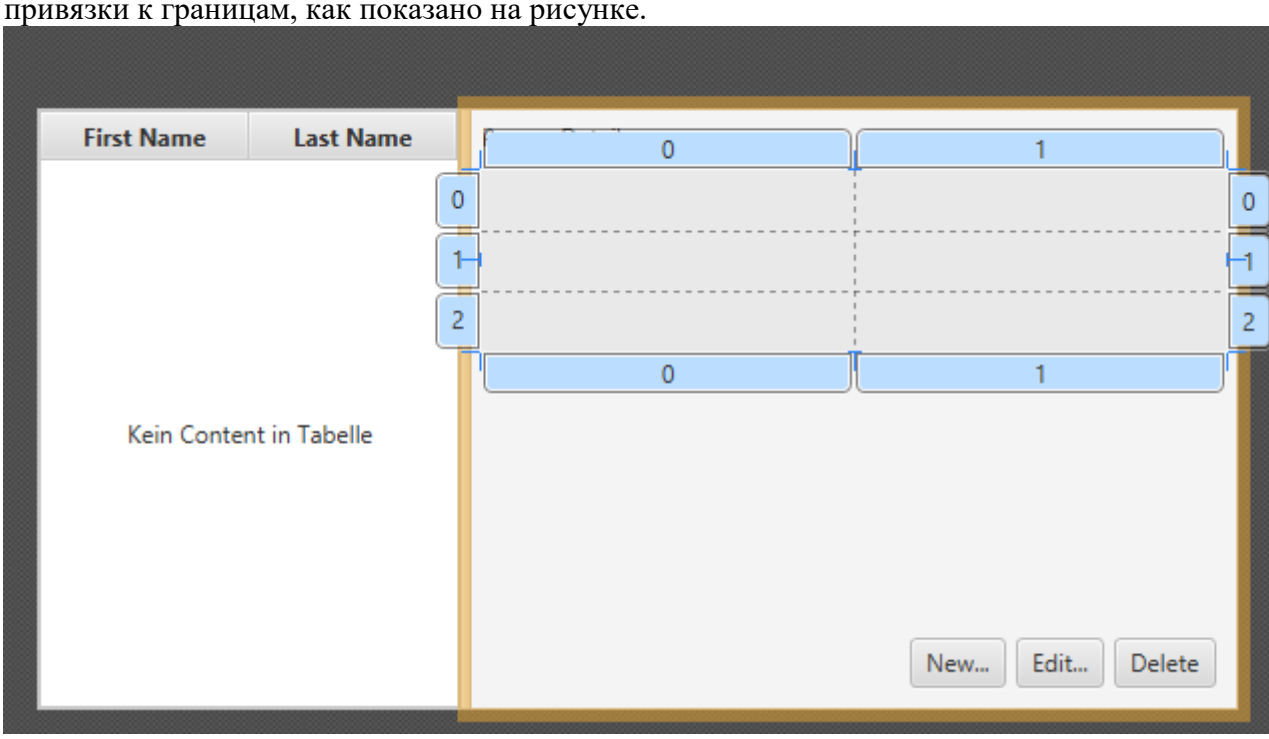
11. Выберите компонент TableView и во вкладке *Properties* измените значение *Column Resize Policy* на *constrained-resize*. Выбор этой характеристики гарантирует, что колонки таблицы всегда будут занимать всё доступное пространство.



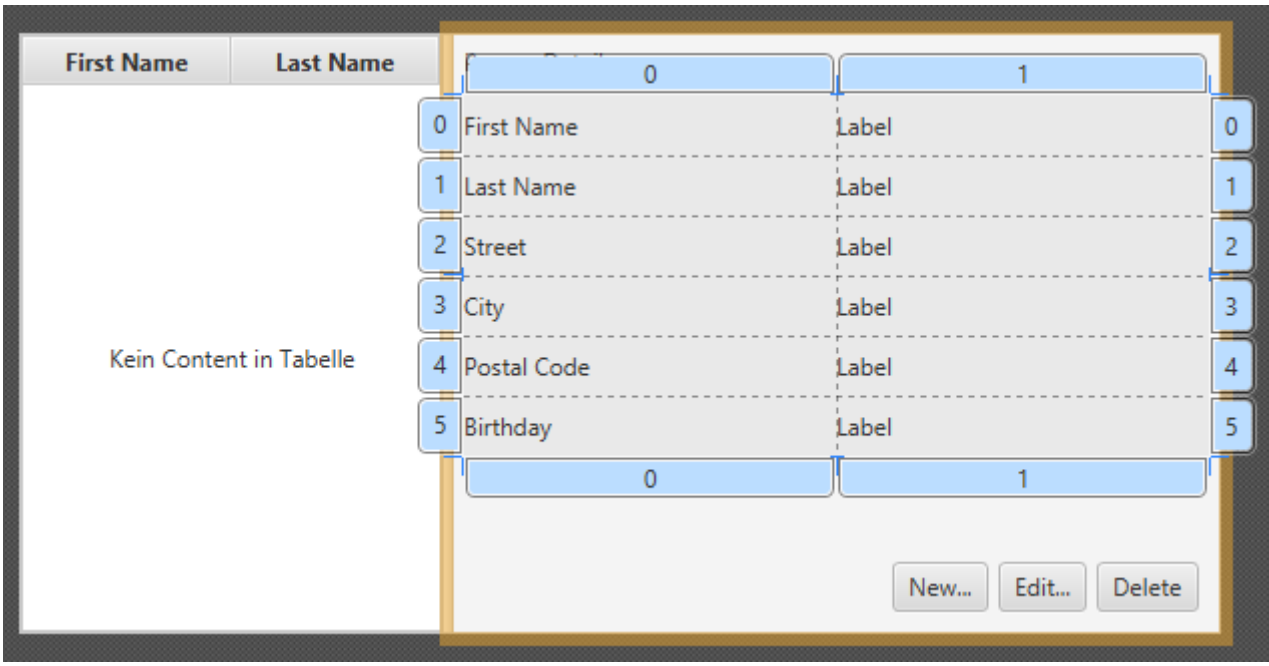
12. На вкладке *Hierarchy* в компонент *SplitPane* перетащите компонент *Label* и измените его текст на "Person Details" (подсказка: используйте поиск для скорейшего нахождения компонентов). Используя привязки к границам (вкладка *Layout*) скорректируйте его положение.



13. На правую панель *SplitPane* добавьте компонент *GridPane* и так же настройте привязки к границам, как показано на рисунке.



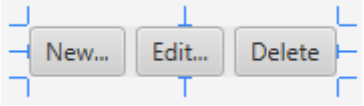
14. Приведите своё окно в соответствие с тем, что показано на рисунке, добавляя компоненты *Label* внутрь ячеек компонента *GridPane*.
Примечание: для того, чтобы добавить новый ряд в компонент GridPane, выберите существующий номер ряда (он окрасится жёлтым), кликните правой кнопкой мышки на номере ряда и выберите пункт "Add Row Above" или "Add Row Below".



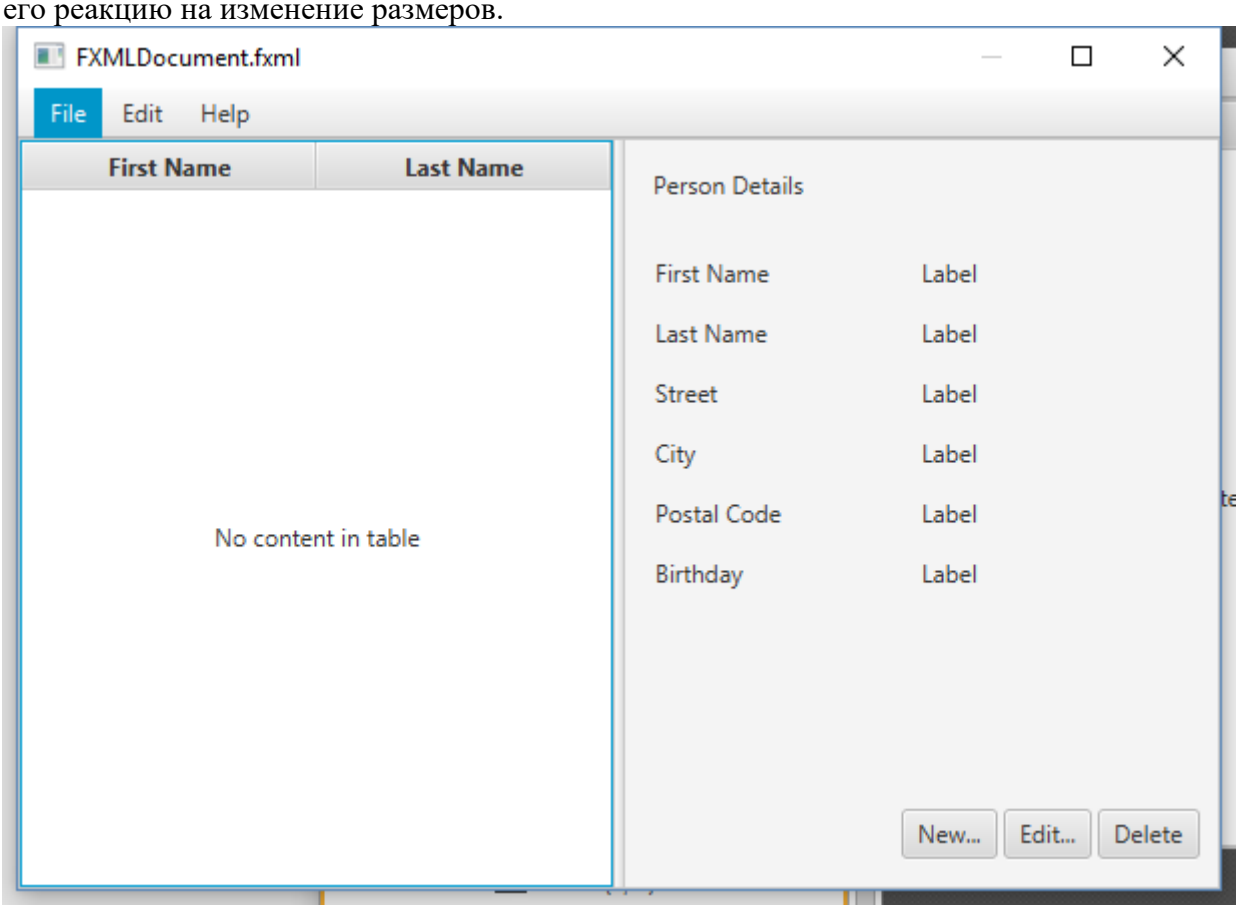
15. Внизу добавьте ButtonBar, а в него три кнопки Button. Теперь установите привязки к границам (правой и нижней), чтобы ButtonBar всегда находилась справа.

Так как панель ButtonBar доступна только с JavaFX 8, и её поддержка в Scene Builder на данный момент несколько хромает, то имеется альтернативный способ. Добавьте три компонента *Button* в правую часть так, как показано на предыдущем рисунке. 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 *Layout* компонента *HBox*. Также установите привязки к границам (правой и нижней).



1. Если всё сделано правильно, то у нас должно получится что-то похожее на рисунок ниже. Используйте пункт меню *Preview*, чтобы протестировать созданное окно и его реакцию на изменение размеров.

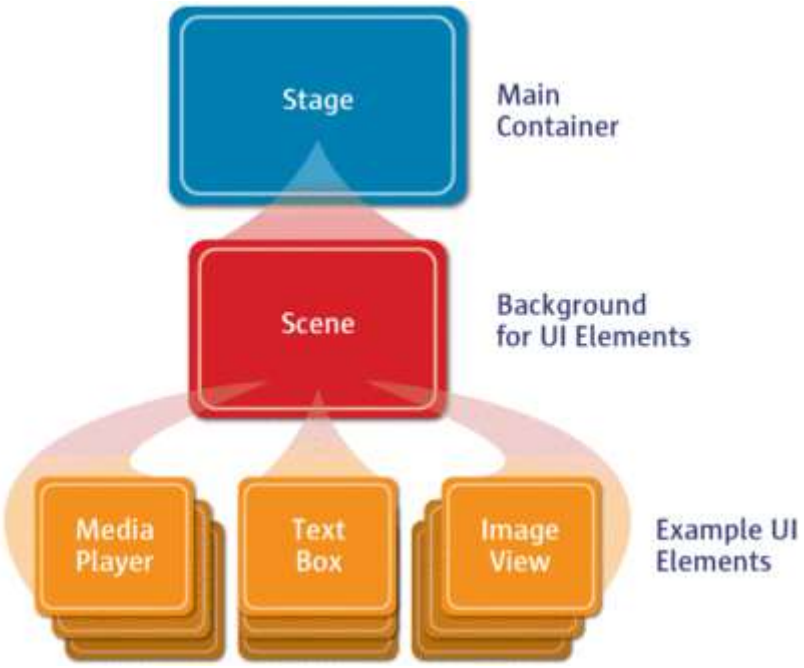


Запуск основного приложения

Основной класс приложения JavaFX

Ранее созданный **основной класс Java (AddressApp.java)**, будет запускать наше приложение с `FXMLDocument.fxml`.

1. На следующем рисунке представлена структура любого приложения JavaFX:



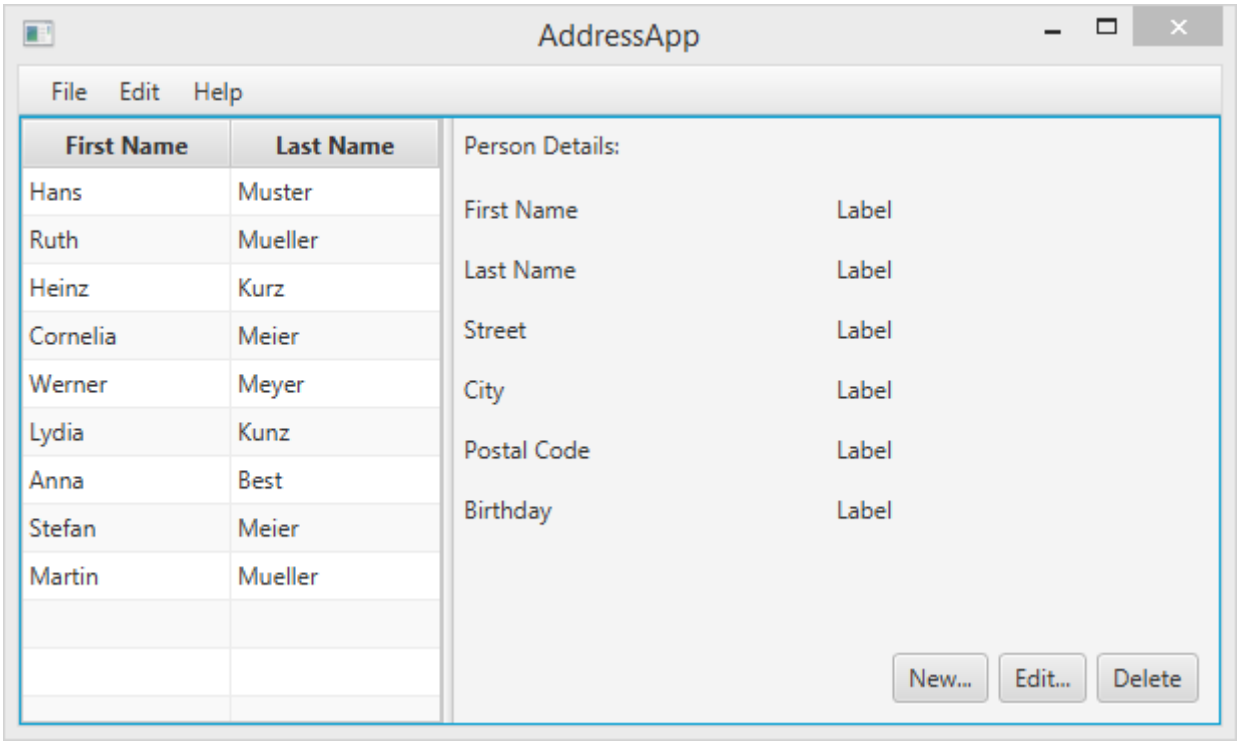
Источник изображения: http://www.oracle.com/

Это как театральное представление `Stage` (театральные подмостки) является основным контейнером, который, как правило, представляет собой обрамлённое окно со стандартными кнопками: закрыть, свернуть, развернуть. Внутрь `Stage` добавляется сцена `Scene`, которая может быть заменена другой `Scene`. Внутрь `Scene` добавляются стандартные компоненты типа `AnchorPane`, `TextBox` и другие.

Для получения более детальной информации о такой компоновке обратитесь к этому руководству: Working with the JavaFX Scene Graph.

Запустив приложение мы должны увидеть что-то похожее на то, что изображено на рисунке в начале этой статьи.

- Часть 2: Модель и компонент TableView



Часть 2: Содержание

- Создание класса-**модели**;
- Использование класса-модели в коллекции **ObservableList**;
- Отображение данных в компоненте **TableView** с помощью **Контроллеров**.

Создание класса-модели

Класс-модель необходим для хранения в нашей будущей адресной книге информации об адресатах. Добавьте класс `Person.java` в пакет `addressapp`. В нём будет несколько переменных для хранения информации об имени, адресе и дне рождения.

Мы так же добавим в код конструктор, который будет создавать некоторые демонстрационный данные и методы-геттер и сеттер с публичным модификатором доступа:

Добавьте в этот класс следующий код.

Person.java

```
package addressapp;

import java.time.LocalDate;

public class Person {

   private  String firstName;
```

```
private  String lastName;
private  String street;
private  Integer postalCode;
private  String city;
private  LocalDate birthday;

public Person(String firstName, String lastName) {
   this.firstName = firstName;
   this.lastName = lastName;
   // Какие-то фиктивные начальные данные для удобства тестирования.
   this.street = "какая-то улица";
   this.postalCode = 1234;
   this.city = "какой-то город";
   this.birthday = LocalDate.of(1999, 2, 21);
}

public String getFirstName() {
   return firstName;
}

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

public String getLastName() {
   return lastName;
}

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

public String getStreet() {
   return street;
}

public void setStreet(String street) {
   this.street = street;
}

public Integer getPostalCode() {
   return postalCode;
}

public void setPostalCode(Integer postalCode) {
   this.postalCode = postalCode;
}

public String getCity() {
   return city;
}

public void setCity(String city) {
   this.city = city;
}

public LocalDate getBirthday() {
   return birthday;
}

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

}

Объяснение

- Класс `LocalDate`, тип которого мы выбрали для нашей переменной `birthday`, это часть нового Date and Time API для JDK 8.

Список людей

Основные данные, которыми оперирует наше приложение - это группа экземпляров класса `Person`. Давайте создадим в классе `FXMLDocumentController.java` список объектов класса `Person`. Все остальные классы-контроллеры позже получат доступ к этому центральному списку внутри этого класса.

Список ObservableList

Мы работаем с классами-представлениями JavaFX, которые необходимо информировать при любых изменениях в списке адресатов. 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 того, чтобы получить доступ к таблице и меткам представления, мы определим некоторые переменные. Эти переменные и некоторые методы имеют специальную аннотацию `@FXML`. Она необходима для того, чтобы fxml-файл имел доступ к приватным полям и методам. После этого мы настроим наш fxml-файл так, что при его загрузке приложение автоматически заполняло эти переменные данными.
- . Итак, давайте добавим следующий код в наш класс:

Примечание: При импорте пакетов всегда используйте пакет ***javafx***, а НЕ *awt* или *swing*!

FXMLDocumentController.java

```
package addressapp;


import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;


public class FXMLDocumentController {
    /**
     * Данные, в виде наблюдаемого списка адресатов.
     */
    private final ObservableList<Person> personData = FXCollections.observableArrayList();
```

```
    @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 streetLabel;
    @FXML
    private Label postalCodeLabel;
    @FXML
    private Label cityLabel;
    @FXML
    private Label birthdayLabel;

    @FXML
    public void initialize() {
        // В качестве образца добавляем некоторые данные
        personData.add(new Person("Hans", "Muster"));
        personData.add(new Person("Ruth", "Mueller"));
        personData.add(new Person("Heinz", "Kurz"));
        personData.add(new Person("Cornelia", "Meier"));
        personData.add(new Person("Werner", "Meyer"));
        personData.add(new Person("Lydia", "Kunz"));
        personData.add(new Person("Anna", "Best"));
        personData.add(new Person("Stefan", "Meier"));
        personData.add(new Person("Martin", "Mueller"));
        // Инициализация таблицы адресатов с двумя столбцами.
        firstNameColumn.setCellValueFactory(new PropertyValueFactory<>("firstName"));
        lastNameColumn.setCellValueFactory(new PropertyValueFactory<>("lastName"));
        // Добавление в таблицу данных из наблюдаемого списка
        personTable.setItems(personData);
    }

}
```

Этот код требует некоторых разъяснений:

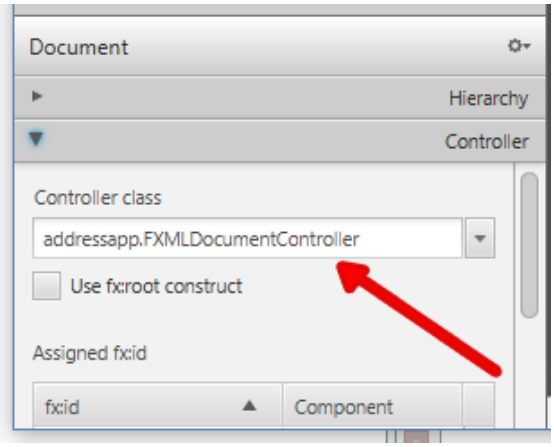
- Все поля и методы, к которым fxml-файлу потребуется доступ, должны быть отмечены аннотацией `@FXML`. Несмотря на то, что это требование предъявляется только для полей и методов с модификатором private, лучше оставить их закрытыми и помечать аннотацией, чем делать публичными!
- После загрузки fxml-файла автоматически вызывается метод `initialize()`. На этот момент все FXML-поля должны быть инициализированы;
- Метод `setCellValueFactory(...)` определяет, какое поле внутри класса `Person` будет использоваться для конкретного столбца в таблице (PropertyValueFactory,).

Класс FXMLDocumentController

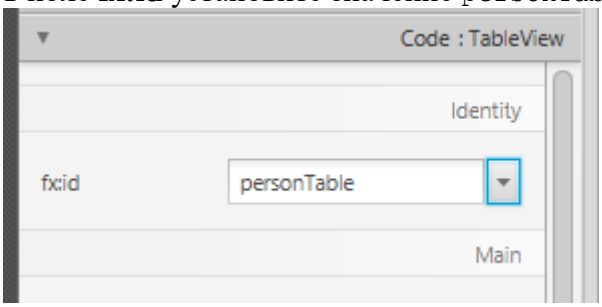
Привязка класса-контроллера к fxml-файлу

Данная часть учебника близится к своему завершению, однако мы пропустили одну маленькую деталь! Мы не сказали файлу `FXMLDocument.fxml`, какой контроллер он должен использовать, а так же не указали соответствие между элементами представления и полями внутри класса-контроллера. Для этого:

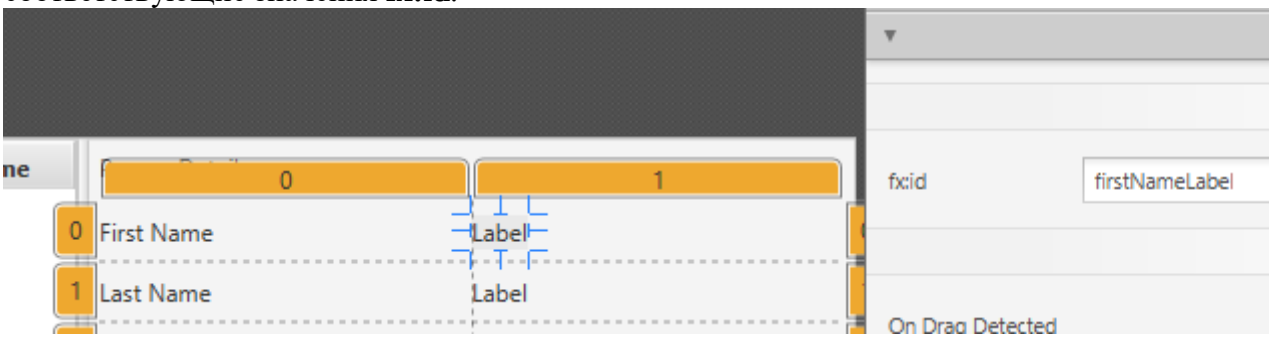
1. Откройте файл `FXMLDocument.fxml` в приложении *Scene Builder*.
2. Откройте вкладку *Controller* слева на панели *Document* и выберите класс `FXMLDocumentController` в качестве класса-контроллера.



3. Выберите компонент `TableView` на вкладке *Hierarchy*, перейдите на вкладку *Code* и в поле **fx:id** установите значение `personTable`.



4. Сделайте то же самое для колонок таблицы и установите значения свойства **fx:id** `firstNameColumn` и `lastNameColumn` соответственно.
5. Для каждой метки во второй колонке компонента GridPane также установите соответствующие значения **fx:id**.



6. Важно: сохраните файл `FXMLDocument.fxml`, вернитесь в среду разработки NetBeans

Запуск приложения

После запуска приложения мы должны увидеть что-то похожее на то, что изображено на картинке в начале данной статьи.

Поздравляю!

Примечание: пока ещё при выборе конкретного адресата у нас не обновляются метки. Взаимодействие с пользователем мы будем программировать в следующей части учебника.