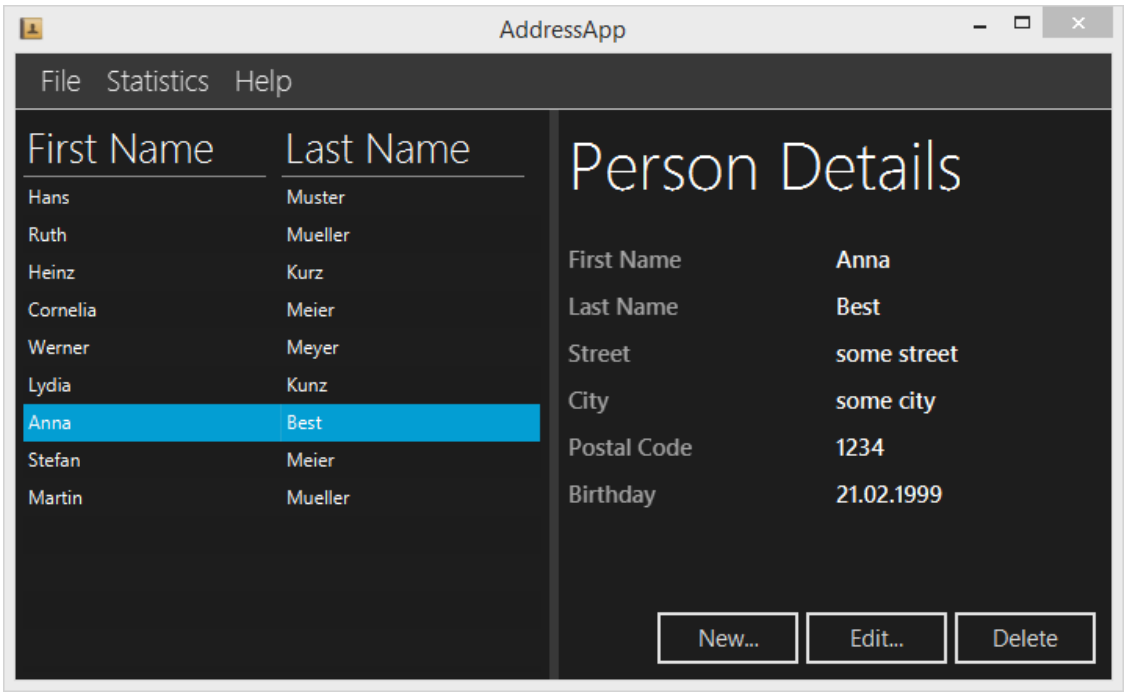
Учебник по JavaFX 8 (Русский)

2014-04-19 00:00

https://github.com/marcojakob/code.makery.ch/tree/master/collections/library

В этом учебнике я расскажу о проектировании, программировании и развертывании приложения с функциональностью адресной книги. Так будет выглядеть наше приложение в конце разработки:



Screenshot AddressApp

Вам предстоит научиться

- Создавать и запускать JavaFX-проект;
- Использовать приложение Scene Builder для проектирования пользовательского интерфейса;
- Структурировать приложение с помощью патерна MVC;
- Использовать коллекцию `ObservableList` для автоматического обновления пользовательского интерфейса;
- Использовать компонент `TableView` и реагировать на выделение ячеек в таблице;
- Создавать пользовательские всплывающие диалоги для редактирования записей в приложении;
- Выполнять проверку пользовательского ввода;
- Изменять дизайн JavaFX-приложения с помощью каскадных таблиц стилей (CSS);
- Хранить данные приложения в виде XML-файла;
- Сохранять последний открытый путь к файлу в пользовательских настройках;

- Создавать JavaFX-диаграммы для отображения статистики;
- Развертывать JavaFX-приложение с помощью процесса нативной упаковки (native package).

Это довольно много! А это значит, что после изучения данного материала вы будете знать, как создавать сложные приложения с помощью JavaFX.

Как пользоваться этим учебником

Есть два варианта использования этого материала:

- **учите много**: Создавайте свой JavaFX проект с нуля и постепенно наполняйте его классы и методы кодом.
- **учите быстро**: Импортируйте исходники кода для каждой части учебника в вашу среду разработки а потом пытайтесь понять данный материал.

Я надеюсь, что вы получите удовольствие! Начнем с Часть 1: Приложение Scene Builder.

Attribution: Russian translations have been contributed by

```
<li><a href="https://github.com/sobolevstp" class="alert-link">Sobolev
Stepan</a></li>
<li><a href="https://github.com/eugenedotru" class="alert-link">Evgen
Ishchenko</a></li>
```

Thank you very much!

Screenshot AddressApp Part 1

Screenshot AddressApp Part 1

Часть 1: Содержание

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

Предпосылки

- Последняя Java JDK 8 (включающая в себя **JavaFX 8**);
- Среда разработки Eclipse версии 4.3 или выше с установленным плагином f(ex)lipse. Сконфигурированную версию среды разработки Eclipse можно скачать с сайта e(fx)lipse. Или использовать сайт обновлений, если Eclipse уже установлена.
- Приложение Scene Builder версии 2.0 или новее.

Настройка среды разработки Eclipse

Нам нужно указать среде разработки Eclipse использовать JDK 8, а также указать путь к приложению Scene Builder:

1. Откройте настройки среды разработки Eclipse и перейдите к пункту *Java* | *Installed JREs*.
2. Нажмите *Add...*, выберите *Standart VM* и укажите путь к установленной JDK 8.
3. Уберите другие добавленные JDK и JDK 8 будет использоваться по умолчанию.
   Preferences JDK
4. Перейдите к пункту *Java* | *Compiler*. Установите значение **Compiler compliance level** на **1.8**.
   Preferences Compliance
5. Перейдите к пункту *JavaFX* и укажите путь к исполняемому файлу приложения Scene Builder.
   Preferences JavaFX

Полезные ссылки

Возможно, вы захотите добавить закладки на следующие ссылки:

- Java 8 API - документация по стандартным классам Java;
- JavaFX 8 API - документация по классам JavaFX;
- ControlsFX API - документация для проекта ControlsFX;
- Oracle's JavaFX Tutorials - официальный учебник по JavaFX от Oracle.

Ну что же, давайте приступим к изучению!

Создание нового JavaFX-проекта

Перейдите в приложение Eclipse и нажмите *File* | *New* | *Other...* и выберите *JavaFX Project*. Укажите имя проекта (например, AdressApp) и нажмите *Finish*.

Если приложение Eclipse автоматически создало начальные файлы и пакеты, то удалите их.

Создание структуры пакетов

С самого начала мы будем следовать хорошим принципам проектирования ПО. Один из них - это патерн MVC. Опираясь на этот паттерн мы разбиваем код нашего приложения на три части и создаем для каждой свой пакет (правык клик на папке src, *New...* | *Package*):

- `ch.makery.adress` - содержит *большинство* классов-контроллеров (Controller) (= бизнес логики);
- `ch.makery.adress.model` - содержит классы Модели (Model);
- `ch.makery.adress.view` - содержит классы Вида (View).

Заметка: Внутри пакета Вид также содержатся некоторые классы-контроллеры, которые непосредственно связаны с одним видом. Давайте назовем их виды-контроллеры (**view-controllers**).

Создание файла разметки FXML

Есть два пути создания пользовательского интерфейса: использовать файл разметки FXML или программировать все на Java. Для большинства случаев мы будем использовать XML (.fxml). Я считаю, этот способ больше подходит для сохранения разделенности Контроллера и Вида друг от друга. В дальнейшем мы сможем использовать Scene Builder для визуального редактирования нашего XML. А это значит, что мы не будем работать с XML на прямую.

Кликните на пакет `view` правуй кнопкой мышки и создайте новый FXML-документ с названием `PersonOverview`.

New FXML Document

New FXML Document

New PersonOverview

New PersonOverview

Проектировка визуального интерфейса в приложении Scene Builder

Заметка: Если по какой-то причине ваш код не работает, скачайте исходники к этой части учебника и попытайтесь открыть скачанный fxml-файл оттуда.

Откройте наш созданный fxml-документ в приложении Scene Builder. На вкладке *Hierarchy* у вас должен быть единственный компонент *AnchorPane*.

1. Выберите компонент *AnchorPane* на вкладке *Hierarchy*, перейдите на вкладку *Layout* и установите значение характеристикам *Pref Width* и *Pref Height* 600 и 300.
   Anchor Pane Size

2. Добавьте компонент *SplitPane (Horizontal Flow)* на вкладку *Hierarchy* на уже добавленный компонент *AnchorPane*. Клацните по нем правой кнопкой миши и выберите *Fit to Parent*.
   Fit to Parent

3. Теперь добавьте компонент TableView (с вкладки *Library*)в левую часть только что добавленного компонента *SplitPane*. Выделите его и проставьте оступы от краев так, как показано на рисунке. Внутри компонента *AnchorPane* вы всегда можете проставить отступы для четырех сторон (дополнительная информация о разметках).
   TableView Anchors

4. Перейдите в меню *Preview | Show Preview in Window* для того, чтобы увидеть правильно ли отображается созданное окно. Попробуйте изменить размер окна. Добавленная таблица должна изменятся вместе с окном, т.к. она прикреплена к границам окна.

5. Измените заголовки колонок в таблице (на вкладке *Properties* компонента *TableColumn*) на “First Name” и “Last Name”.
   Column Texts

6. Выберите наш компонент TableView и измените значение *Column Resize Policy* (на вкладке *Properties*) на *‘constrained-resize’*. Выбор этой характеристики гарантирует, что колонки таблицы всегда будут занимать все доступное пространство.
   Column Resize Policy

7. Добавьте компонент *Label* на правую часть компонента *SplitPane* и измените его текст на “Person Details” (подсказка: для нахождения компонентов вы можете использовать поиск). Скоректируйте его положение используя привязки к границам (на вкладке *Layout*).
   Person Details Label

8. Добавьте компонент *GridPane* на правую и тоже настройте привязки к границам так, как показано на рисунке.
   GridPane Layout

9. Приведите свое окно в соответствие с тем, что показано на рисунке, добавляя компоненты *Label* внутрь ячеек компонента *GridPane*. Для того, чтобы добавить новый ряд в компонент *GridPane*, кликните правой кнопкой мышки на номере рядка и выберите пункт “*Add Row*”.
   Add labels

10. Добавьте три компонента *Button* на правую часть так, как показано на предыдущем рисунке. Выделите их всех вместе, кликните по ним правой клавишой мышки и выберите пункт *Wrap In | HBox*. Это действие их сгруппирует. Вы можете задать расстояние между компонентами во вкладке *Properties* компонента *HBox*. Также установите привязки к границам (правой и нижней).
    Button Group

11. Если вы все сделали правильно, то у вас должно получится что-то похожее на это. Для того, чтобы протестировать созданное окно используйте пункт меню *Preview*.
    Preview

Создание основного приложения

Нам необходимо создать еще один файл fxml-разметки, в котором будет компонент *Menu Bar* и который будет служить оберткой для только что созданного `PersonOverview.fxml`.

1. Создайте другой fxml-файл в том же пакете, что и предыдущий и назовите его `RootLayout.fxml`.
   New RootLayout

2. Откройте файл `RootLayout.fxml` в приложении Scene Builder.

3. Установите предпочитаемое значение ширины и высоты компонента 600 и 400 соответственно.
   RootLayout Size

4. Добавьте компонент *MenuBar* в верхний слот компонента *BorderPane*. Функциональность меню мы будем реализовывать чуть позже.
   MenuBar

Основной класс JavaFX-приложения

Теперь нам надо создать основной класс, который запускает наше приложение с `RootLayout.fxml` и добавляет `PersonOverview.fxml` в центр.

1. Кликните правой кнопкой мыши по вашему проекту и перейдите на *New* | *Other...* и выберите *JavaFX Main Class*. New JavaFX Main Class

2. Назовите класс `MainApp` и поместите его в пакет `ch.makery.address` (заметка: это родительский пакет таких пакетов как `view` и `model`). New JavaFX Main Class

Созданный класс `MainApp.java` расширяет класс `Application` и содержит два метода. Это базовая структура для запуска JavaFX-приложения. Для нас важен метод `start(Stage primaryStage)`. Он автоматически вызывается при вызове метода `launch(...)` с метода `main`.

Как вы видите, метод `start(...)` принимает экземпляр класса `Stage` в роли параметра. На рисунке снизу представлена структура любого JavaFX-приложения:

New FXML Document *Источник изображения: http://www.oracle.com/*

`Stage` является основным контейнером, который, как правило, представляет из себя окно с рамками и стандартными кнопками закрыть, уменьшить и увеличить. Внутрь `Stage` добавляется `Scene`, которая может быть заменена другой `Scene`. Внутрь `Scene` уже добавляются стандартные компоненты типа `AnchorPane`, `TextBox` и другие.

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

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

Комментарии могут служить вам в роли подсказок того, что и как делается.

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

Частые проблемы

Если приложение не сможет найти указанного `fxml`-файла, вы получите следующее предупреждение:

```
java.lang.IllegalStateException: Location is not set.
```

Для решения данной проблемы проверьте правильность указания пути к файлу и правильность написания его названия.

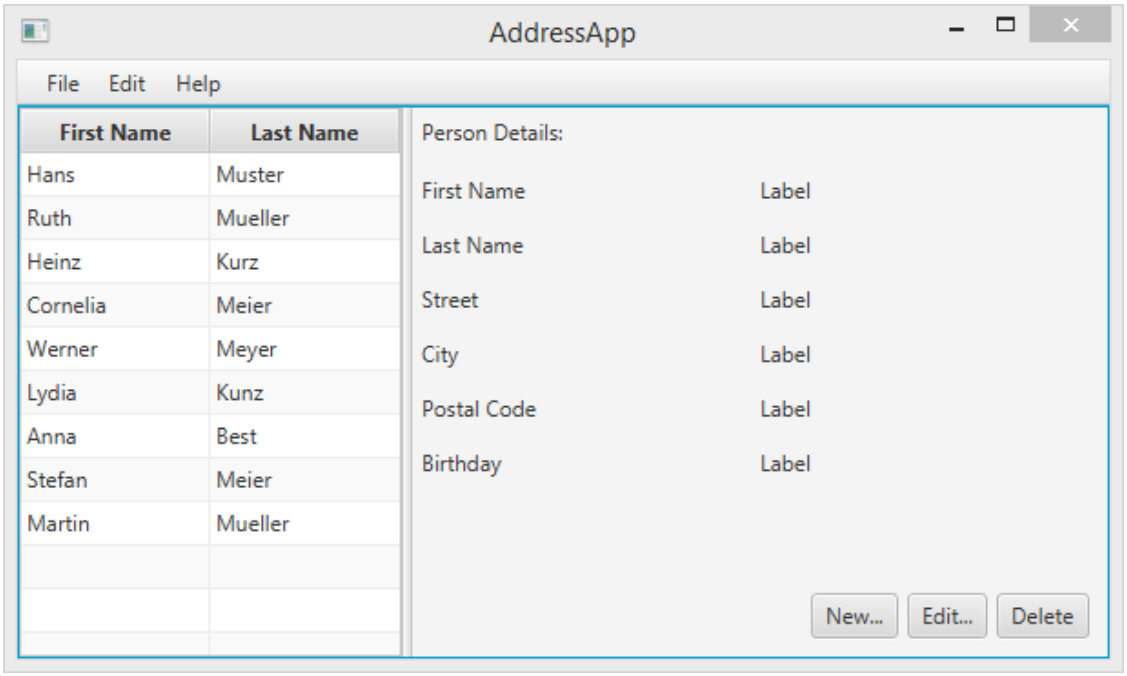
Если после этого вам все же не удастся запустить приложение, попробуйте скачать исходники к этой части и запустить их.

Что дальше?

Во 2 Части Учебника мы добавим в наше приложение некоторые данные и функциональность.

Вам могут быть интересны также некоторые другие статьи

- JavaFX Dialogs
- JavaFX Date Picker
- JavaFX Event Handling Examples
- JavaFX TableView Sorting and Filtering
- JavaFX TableView Cell Renderer



Screenshot AddressApp Part 2

Часть 2: Содержание

- Создание класса-**модели**;
- Использование класса-модели в коллекции **ObservableList**;
- Отображение данных в компоненте **TableView** используя **Контроллеры**.

Создание класса-модели

Класс-модель нам необходим для хранения информации о людях в нашей адресной книге. Добавьте класс `Person.java` в пакет `ch.makery.address.model`. В нем будет несколько переменных для хранения имени, адреса и дня рождения. Добавьте следующий код в этот класс.

Person.java

Объяснения

- В JavaFX предпочтительно использовать Properties для всех полей класса-модели. `Property` позволяет нам получать автоматические уведомления при любых изменениях переменных, таких как `lastName` или других. Это помогает нам сохранять синхронность отображения и данных. Для более делатьного изучения `Properties` читайте Using JavaFX Properties and Binding;
- Класс LocalDate, тип которого мы выбрали для нашей переменной `birthday`, это часть нового Date and Time API для JDK 8.

Список записей

Основные данные, которыми оперирует наше приложение - это связка экземпляров класса `Person`. Давайте создадим список объектов класса `Person` внутри класса `MainApp.java`. Все остальные классы-контроллеры позже получат доступ к этой центральной коллекции внутри класса `MainApp.java`.

Список ObservableList

Мы работаем с классами-видами, которые необходимо информировать при любых изменениях в записях адресной книги. Не будь этого, мы бы не могли синхронизировать отображение данных с самими данными, которые у нас есть. Для этой цели в JavaFX предоставлены некоторые новые классы-коллекции.

Из этих классов нам необходим класс `ObservableList`. Для создания экземпляра данного класса добавьте приведенный код в начало нашего класса `MainApp.java`. Также добавьте в код конструктор, внутри которого наш список будет наполняться данными и добавьте геттер для нашего списка с публичным модификатором доступа:

MainApp.java

Класс PersonOverviewController

Теперь мы отобразим некоторые данные в нашей таблице. Для этого нам необходимо создать класс-контроллер для нашего `PersonOverview.fxml`.

1. Создайте новый класс внутри пакета `view` и назовите его `PersonOverviewController.java`. (Мы должны разместить наш класс-контроллер в

тот же пакет, где находится наш файл разметки `PersonOverview.fxml`, иначе приложение Scene Builder не сможет найти его - по крайней мере не в текущей версии);

2. Для того, чтобы получить доступ к таблице и меткам внутри нашего окна, 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 приватным методам. После этого мы настроем наш fxml-файл так, что при его загрузке приложение автоматически будет заполнять эти переменные данными. Ну что, давайте добавим следующий код в наш класс:

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

PersonOverviewController.java

Наш код требует некоторых объяснений:

- Все поля и методы, к которым fxml-файлу потребуется доступ, должны быть помечены аннотацией `@FXML`. Правда, это верно только для полей и методов с модификатором private, но лучше оставить их такими и помечать аннотацией, чем делать публичными! Метод `initialize()` автоматически вызывается после загрузки fxml-файла. На этот момент все FXML-поля должны быть иниализированны; * Метод `setCellValueFactory(...)` используется для определения того, какое поле внутри класса `Person` будут использоватся 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).

Соединение класса MainApp с классом PersonOverviewController

Метод `setMainApp(...)` должен быть вызван с 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()` две дополнительные строки:

MainApp.java - метод showPersonOverview()

Привязка класса-Контроллера к fxml-файлу

Мы подходим к завершению данной части! Но мы пропустили одну маленькую вещь: мы не сказали нашему файлу `PersonOverview.fxml`, какой контроллер он должен использовать и какие элементы вида каким полям внутри класса-контроллера должны соответствовать. Для этого:

1. Откройте файл `PersonOverview.fxml` в приложении *Scene Builder*.

2. Откройте вкладку *Controller* слева на панели *Document* и выберите класс `PersonControllerOverview` в качестве класса-контроллера.
Set Controller Class

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

4. Сделайте то же самое для колонок таблицы и установите значения свойства **fx:id** `firstNameColumn` и `secondNameColumn` соответственно.

5. Для каждой метки во второй колонке компонента GridPane также установите соответствующие значения **fx:id**.
   Set Label fx:id

6. Важно: сохранение наш файл `PersonOverview.fxml`, а потом вернитесь в среду разработки Eclipse и обновите весь проект AdressApp (F5). Это необходимо для того, чтобы приложение Eclipse определило те изменения, которые мы сделали в приложении Scene Builder.

Запуск приложения

Когда вы запустите ваше приложение, вы должны будете увидеть что-то похожее на то, что изображено на картинке вначале данной статьи.

Поздравляю!

Что дальше?

В 3 Части Учебника мы научим наше приложение добавлять, редактировать и удалять адресные записи.

Вам могут быть интересны также некоторые другие статьи

- JavaFX Dialogs
- JavaFX Date Picker
- JavaFX Event Handling Examples
- JavaFX TableView Sorting and Filtering
- JavaFX TableView Cell Renderer

Учебник по JavaFX 8 - Часть 3: Взаимодействие с пользователем

Учебник по JavaFX 8 - Часть 3: Взаимодействие с пользователем

Часть 3: Содержание

- Реакция на выбор записей в таблице.
- Добавление функциональности к кнопкам **add**, **edit** и **remove**.
- Создание пользовательского диалогового окна для редактирование записей адресной книги.
- Проверка пользовательского ввода.

Реакция на выбор записей в таблице

Пока мы еще не использовали правую часть нашего приложения. Идея заключается в том, чтобы при выборе записи в таблице отображать детали этой записи в правой части приложения.

Сначала давайте добавим новый метод в класс `PersonOverviewController` который поможет нам заполнять текстовые метки данными указанной записи.

Создайте метод `showPersonDetails(Person person)`. Пройдитесь по меткам и присвойте им соответствующие значения, взятые из переданной в параметре записи, используя метод `setText(...)`. Если в качестве параметра передается `null`, весь текст в метках должен быть очищен.

PersonOverviewController.java

Конвертация дня рождения в строку

Обратите внимание, что мы не можем присвоить значение поля `birthday` тектовой метке, т.к. тип этого значения `LocalDate` а не `String`. Для того чтобы это сделать, нам надо переформатировать нашу дату рождения.

Т.к. мы будем использовать конвертацию типа `LocalDate` в `String` и наоборот в нескольких местах, то хорошей практикой считается создание класса-помощника, содержащего статические методы для этой цели. Наш класс-помощник мы назовем `DateUtil` и разместим его в новый пакет `util`:

DateUtil.java

Подсказка: Вы можете изменить формат даты изменяя константу `DATE_PATTERN`. За всеми возможными форматами смотрите документацию к классу DateTimeFormatter

Использование класса DateUtil

Теперь нам необходимо использоавть наш новый класс `DateUtil` в методе `showPersonDetails` класса `PersonOverviewController`. Замените метку *TODO* следующей строкой:

Следим за изменением выбора записи в таблице

Для получения информации о том, что пользователь выбрал запись в таблице нам необходимо *прослушивать изменения*.

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

Мы будем создавать интерфейс `ChangeListener` пользуясь *лямбда-выражениями* из Java 8. Давайте добавим несколько строчек кода к методу `initialize()` класса `PersonOverviewController`. Теперь наш метод выглядит так:

PersonOverviewController.java

Если мы передаем `null` в параметр метода `showPersonDetails(...)`, то будут стерты все значения меток.

В строке `personTable.getSelectionModel...` мы получаем *selectedItemProperty* таблицы и добавляем к нему слушателя. Когда пользователь выбирает запись в таблице выполняется наше лямбда-выражение. Мы берем только что выбранную запись и передаем ее в метод `showPersonDetails(...)`.

Запустите свое приложение и проверьте отображаются ли детали записи в правой части когда вы выбираете запись из таблицы.

Если у вас что-то не работает, то вы можете сравнить свой класс `PersonOverviewController` с PersonOverviewController.java.

Кнопка Delete

В нашем интерфейсе пользователя есть кнопка **Delete**, но пока у нее нет функциональности. В приложении Scene Builder мы можем указать на то, какое действие будет выполняться при нажатии на эту кнопку. Любой метод внутри класса-контроллера помеченный аннотацией @FXML (или публичный) доступен приложению Scene Builder. Поэтому, давайте сперва добавим метод удаления записи в конец нашего класса `PersonOverviewController`, а потом уже назначим его в роли обработчика кнопки **Delete**.

PersonOverviewController.java

Теперь откройте файл `PersonOverview.fxml` в приложении Scene Builder. Выберите кнопку **Delete**, откройте вкладку *Code* и проставьте метод `handleDeletePerson` в значение пункта *On Action*.

On Action

On Action

Обработка ошибок

Если вы сейчас запустите наше приложение, то вы сможете удалять выбранную запись из таблицы. Но что случится, когда вы нажмете кнопку **Delete**, а ни одна запись выбрана не будет?

Нам выбросит исключение `ArrayIndexOutOfBoundsException`, потому что мы не можем удалить запись с индексом `-1`. Индекс `-1` возвращается методом `getSelectedIndex()` когда в таблице не выделено ни одной ячейки.

Конечно, не очень хорошо игнорировать такую ошибку. Поэтому мы должны сообщить пользователю что он, перед тем как нажимать кнопку **Delete**, должен выбрать запись в таблице.

Для оповещения пользователя мы будем показывать всплывающее диалоговое окно. Для этого вам нужно добавить стороннюю библиотеку (Dialogs):

1. Скачайте этот файл controlsfx-8.0.6_20.jar (вы также можете его скачать с сайта http://fxexperience.com/controlsfx/);
   Важно: Версия библиотеки ControlFX должна быть не ниже `8.0.6\_20` работающая с `JDK 8u20` или выше, т.к. в этой версии были представлены критические изменения.
2. Создайте папку **lib** в корневой папке проекта и добавьте туда файл библиотеки `controlsfx-jar`;
3. Подключите данную библиотеку в ваш проект: В среде разработки Eclipse кликните правой кнопкой мышки на jar-файл | Build Path | Add to Build Path. Теперь приложение Eclipse знает об этой библиотеке.

ControlsFX Libaray

ControlsFX Libaray

Добавив некоторые изменения в метод `handleDeletePerson` мы можем показывать простое диалоговое окно когда пользователь нажимает на кнопку **Delete**, а в таблице ничего не выбрано:

PersonOverviewController.java

Чтобы посмотреть дополнительные примеры использования Диалогов, читайте статью Диалоги в JavaFX 8

Диалоги создания и редактирования записей

Для реализации методов-обработчиков создания и редактирования записей нам потребуется потрудится немного больше. Нам необходимо создать новое пользовательское окно (сцену) с формой, содержащей поля для заполнения всех деталей записи.

Дизайн Окна Редактирования

1. Создайте новый fxml-файл `PersonEditDialog` внутри пакета `view`.
   Create Edit Dialog
2. Используйте компоненты `GridPane`, `Label`, `TextField` и `Button` для создания окна редактирования, похожего на это:
   Edit Dialog

Если что-то не работает, вы можете скачайть PersonEditDialog.fxml.

Создание Контроллера

Создайте класс-контроллер `PersonEditDialogController.java`:

PersonEditDialogController.java

Некоторые заметки по поводу этого контроллера:

- Метод `setPerson(...)` может быть вызван с другого класса для сохранения записи, которая была отредактирована;
- Когда пользователь нажимает на кнопку **OK**, то вызывается метод `handleOK()`. Сперва идет проверка введенных пользователем данных посредством вызова метода `isInputValid()`. Если проверка прошла успешно, идет заполнение записи теми данными, которые ввел пользователь. Эти изменения будут напрямую примняться к записи, принятой в методе `setPerson(...)`!
- Логическая переменная `okClicked` используется для определения того, какую из кнопок `OK` или `Cancel` пользователь нажал.

Привязка класса-контроллера к fxml-файлу

Для работоспособности нашей программы мы должны связать вместе наш класс-контроллер с fxml-файлом. Для этого выполните следующие действия:

1. Откройте файл `PersonEditDialog.fxml` в приложении Scene Builder;
2. С левой стороны во вкладке *Controller* установите наш класс `PersonEditDialogController` как значение параметра *Controller Class*;
3. Установине соответствующие значения параметра **fx:id** для всех компонентов `TextField`;
4. Установите значение параметра **onAction** для наших кнопок и присвойте им соответствующие методы-обработчики.

Вызов диалога редактирования

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

MainApp.java

Добавьте следующие методы в класс `PersonOverviewController`. Когда пользователь будет нажимать на кнопки `New...` или `Edit...` эти методы будут вызывать метод `showPersonEditDialog(...)` из класса `MainApp`.

PersonOverviewController.java

Откройте `PersonOverview.fxml` в приложении Scene Builder и присвойте соответствующие методы-обработчики параметру *On Action* кнопок `New...` и `Edit...`.

Сделано!

Сейчас вы должны получить работающее приложение адресной книги. Приложение способно добавлять, редактировать и удалять записи. Также оно способно делать проверку пользовательского ввода.

Я надеюсь, что концепция и структура даного приложения поможет вам начать писать свои собственные JavaFX-приложения! Удачи.

Что дальше?

В 4 Части Учебника мы будем подключать CSS-стили к нашему приложению.

Вам могут быть интересны также некоторые другие статьи

- JavaFX Dialogs
- JavaFX Date Picker
- JavaFX Event Handling Examples
- JavaFX TableView Sorting and Filtering
- JavaFX TableView Cell Renderer

Screenshot AddressApp Part 4

Screenshot AddressApp Part 4

Часть 4: Содержание

- Стилизация с помощью каскадных таблиц стилей (CSS)
- Добавление иконки приложения

Стилизация с помощью CSS

В JavaFX вы можете стилизировать свой интерфейс пользователя с помощью каскадных таблиц стилей (CSS). Это очень хорошо! Еще никогда не было так легко настроить внешний вид Java-приложения.

В этом учебнике мы создадим тему *DarkTheme*, вдохновленную Метро-дизайном из Windows 8. Используемые стили для кнопок базируются на статье JMetro - Windows 8 Metro controls on Java, написанной Pedro Duque Vieira.

Знакомство с CSS

Если вы хотите стилизировать ваше JavaFX-приложение, вы должны иметь начальное представление о CSS в целом. Хорошее место для старта - этот учебник CSS.

Для получения специфической информации про использование CSS в JavaFX читайте это:

- Skinning JavaFX Applications with CSS - учебник от Oracle
- JavaFX CSS Reference - официальный справочник

Стили, используемые в JavaFX по умолчанию

Стиль, который используется в JavaFX по умолчанию хранится в файле `modena.css`. Этот css-файл можно найти в файле `jfxrt.jar`, расположенном в вашей Java-директории `/jdk1.8.x/jre/lib/ext/jfxrt.jar`.

Разархивируйте его и вы найдете `modena.css` в папке `com/sun/javafx/scene/control/skin/modena`.

Этот стиль всегда применяется для JavaFX-приложений по умолчанию. Добавляя пользовательские стили вы переопределяете стили из файла `modena.css`.

Намек: Для того, чтобы знать какие стили вам надо переопределить, просмотрите этот файл.

Установка пользовательських стилей

Добавьте файл `DarkTheme.css` в пакет `view`.

DarkTheme.css

Теперь нам надо присоединить эти стили к нашей сцене. Можно сделать это программно в коде Java, но мы для того, чтобы добавить стили в наши fxml-файлы будем использовать приложение Scene Builder:

Присоединяем таблицы стилей к файлу RootLayout.fxml

1. Откройте файл `RootLayout.fxml` в приложении Scene Builder.
2. Выберите корневой контейнер `BorderPane` во вкладке *Hierarchy*, перейдите на вкладку *Properties* и укажите файл `DarkTheme.css` в роли таблиц стилей. DarkTheme for RootLayout

Присоединяем таблицы стилей к файлу PersonEditDialog.fxml

1. Откройте файл `PersonEditDialog.fxml` в приложении Scene Builder. Выберите корневой контейнер `AnchorPane` во вкладке *Hierarchy*, перейдите на вкладку *Properties* и укажите файл `DarkTheme.css` в роли таблиц стилей.
2. Фон остался белым, поэтому укажите для корневого компонента `AnchorPane` в классе стиля значение `background`. Add Style Class
3. Выберите кнопку *OK* и отметьте свойство *Default Button* в вкладке *Properties*. В результате изменется цвет кнопки и наша кнопка будет использоваться по умолчанию когда пользователь находясь в окне будет нажимать клавишу enter.

Присоединяем таблицы стилей к файлу PersonOverview.fxml

1. Откройте файл `PersonOverview.fxml` в приложении Scene Builder. Выберите корневой контейнер `AnchorPane` во вкладке *Hierarchy*, перейдите на вкладку *Properties* и укажите файл `DarkTheme.css` в роли таблиц стилей.
2. Вы сражу должны увидеть некоторые изменения: цвет таблицы и кнопок стал черным. Все классы стилей `.table-view` и `.button` из файла `modena.css` применились к таблице и кнопкам. С того момента, как мы переопределили некоторые из стилей в нашем css-файле, новые стили применяются автоматически.
3. Возможно, вам потребуется изменить размер кнопок для того, чтобы отображался весь текст.

4. Выберите правый компонент `AnchorPane` внутри компонента `SplitPane`.
   Background Style Select

5. Перейдите на вкладку *Properties* и укажине в классе стиля значение `background`.
   Теперь фон станет черного цвета.
   Background Style

Текстовые метки с другими стилями

Теперь все наши текстовые метки с правой стороны имеют одинаковый размер. Для дальнейшей стилизации тектсовых меток мы будем использовать уже определенные стили `.label-header` и `label-bright`.

1. Выберите метку *Person Details* и добавьте в качестве класса стиля значение `label-header`.
   Label Header Style

2. Для каждой метки в правой колонке (где отображаются актуальные данные наших записей) добавьте в качестве класса стиля значение `label-bright`.
   Label Bright Style

Добавляем иконку приложения

На данный момент в нашем приложении в панели названия и панели задач используется иконка по умолчанию:

Default Icon

Default Icon

С пользовательской иконкой наше приложение будет выглядеть красивее:

Custom Icon

Custom Icon

Файл изображения

Одно из возможных мест, где можно свободно скачать иконки - это Icon Finder. Я загрузил малельнькую иконку адресной книги.

Создайте папку **resources** внутри вашего проекта AddressApp, а в ней папку **images**. Поместите выбранную вами иконку в папку изображений. Ваша структура папок должна иметь такой вид:

Custom Icon File

Custom Icon File

Установка иконки в приложение

Для установки выбранной иконки в наше приложение добавьте следующий код в метод `start(...)` в классе `MainApp.java`

MainApp.java

The whole `start(...)` method should look something like this now:

Вы также можете добавить иконку в окно редактирования адресной записи.

Что дальше?

В 5 Части Учебника мы добавим XML-хранилище для наших данных.

Вам могут быть интересны также некоторые другие статьи

- JavaFX Dialogs
- JavaFX Date Picker
- JavaFX Event Handling Examples
- JavaFX TableView Sorting and Filtering
- JavaFX TableView Cell Renderer

Screenshot AddressApp Part 5

Screenshot AddressApp Part 5

Часть 5: Содержание

- **Хранение данных в XML**
- Использование компонента JavaFX **FileChooser**
- Использование компонента JavaFX **Menu**
- Сохранение пути к последнему открытому файлу в пользовательских настройках

На данный момент наше приложение умеет хранить данные только в памяти. Каждый раз, когда мы закрываем приложение наши данные теряются. Поэтому настало время подумать о постоянном хранении данных.

Сохранение пользовательских настроек

Java позволяет нам хранить некоторые данные о состоянии приложения в классе `Preferences`. В зависимости от операционной системы, класс `Preferences` записывает данные в разные места (например, в файл регистра в ОС Windows).

Мы не можем использовать класс `Preferences` для хранения всей нашей записной книжки. Но мы можем сохранить там некоторые настройки приложения, например, путь к последнему открытому файлу. Имея эти данные, мы сможем восстанавливать состояние нашего приложения после перезагрузки.

Следующие два метода обеспечивают сохранение и восстановление настроек нашего приложения. Добавьте их в конец класса `MainApp`:

MainApp.java

Хранение данных в XML

Почему именно XML?

Использование баз данных является одним из наиболее распространенных способов хранения данных. Пока наши данные, которые мы должны хранить, являются объектами, базы данных содержат их в виде реляционных данных (например, таблиц). Это называется object-relational impedance mismatch. Но для того, чтобы привести наши объектные данные в соответствие с реалицяонными таблицами требуется выполнить дополнительную работу. Существуют фреймворки, которые помогают приводить наши объектные данные в соответствие с реляционной базой данных (Hibernate - один из наиболее популярных), но чтобы начать их использовать, также необходимо проделать дополнительную работу на настройку.

Для нашего простого приложения намного легче хранить данные в виде XML. Для этого мы будем использовать библиотеку JAXB (**J**ava **A**rchitechture for **X**ML **B**inding). Написав всего несколько строк кода, JAXB позволит нам сгенерировать такой исходный XML-файл:

Пример сгенерированного XML-файла

Использование JAXB

Библиотека JAXB изначально включена в JDK. Это значит, что никаких дополнительных библиотек нам подключать не придется.

JAXB предоставляет две основные функции: способность к **маршаллированию** Java-объектов в XML и обратную **демаршализацию** с xml-файла в Java-объекты.

Для того, чтобы JAXB предоставила возможность такой конвертации, нам необходимо подготовить нашу модель.

Подготовка класса-модели для JAXB

Данные, которые мы хотим сохранить находятся в переменной `personData` в классе `MainApp`. JAXB требует пометить аннотацией `@XmlRootElement` внешний класс наших данных (только класс, поле этой аннотацией пометить нельзя). Тип переменной `personData` является `ObservableList`, на который мы не можем поставить аннотацию. Поэтому, для того, чтобы решить эту проблему нам необходимо создать класс-обертку, который будет содержать только нашу коллекцию записей, и на который мы поставим аннотацию `@XmlRootElement`.

Создайте в пакете `model` новый класс `PersonListWrapper`.

PersonListWrapper.java

Обратите внимание на две аннотации:

- `@XmlRootElement` определяет имя корневого элемента.
- `@XmlElement` оппределяет имя элемента, которое нам указывать необязательно.

Чтение и запись данных с помощью JAXB

Сделаем наш класс `MainApp` ответственным за чтение и запись данных нашего приложения. Для этого добавьте два метода в конец класса `MainApp.java`:

Возможность маршаллинга и демаршализации готовы. Теперь давайте создадим пункты меню "*Сохранить*" и "*Загрузить*" для того, чтобы использовать данные возможности.

Обработка действий меню

Мы уже создавали меню в файле `RootLayout.fxml`, которое пока еще не использовали. Перед тем, как мы добавим поведение нашему меню, давайте создадим все необходимые нам пункты.

Откройте файл `RootLayout.fxml` в приложении Scene Builder и перенесите необходимое количество пунктов меню из вкладки *Library* на вкладку *Hierarchy*. Создайте следующие пункты меню: **New**, **Open...**, **Save**, **Save as...** и **Exit**.

Add Menu Items

Add Menu Items

Подсказка: Используйте свойство *Accelerator* в вкладке *Properties* для установки горячих клавиш на пункты меню.

Класс RootLayoutController

Для обработки поведения меню нам необходимо новый класс-контроллер. Создайте класс `RootLayoutController` в пакете `ch.makery.address.view`.

Добавьте к классу-контроллеру следующее содержание:

RootLayoutController.java

Компонент FileChooser

Обратите внимание на методы, которые используют компонент `FileChooser` в классе `RootLayoutController`. Сперва мы создаем новый экземпляр класса `FileChooser`. Потом мы применяем фильтр расширешия и при выборе файлов будут показываться только те, которые имеют расширение `.xml`. Ну и наконец, мы отображаем данный компонент поверх нашей *PrimaryStage*.

Если пользователь закрывает диалог выбора файлов не выбрав ни никакого файла, тогда возвращается `null`. В противном случае мы берем выбранный файл и передаем его внутрь метода `loadPersonDataFromFile(...)` или `savePersonDataToFile(...)`, которые находятся в классе `MainApp.`

Соединение fxml-файла с классом-контроллером

1. Откройте файл `RootLayout.fxml` в приложении Scene Builder. На вкладке *Controller* в качестве класса-контроллера выберите значение `RootLayoutController`.

2. Перейдите на вкладку *Hierarchy* и выберите пункт меню. Во вкладке *Code* в качестве значений свойства *On Action* вы сможете увидеть все доступные методы выбранного класса-контроллера. Выберите метод, соответствующий данному пункту меню.
   Menu Actions

3. Повторите предыдущий шаг для каждого пункта меню.

4. Закройте приложение Scene Builder и оновите ваш проект (нажмите **Refresh (F5)** на корневой папке вашего проекта). Это позволит среде разработки Eclipse увидеть сделанные вами изменения в приложении Scene Builder.

Соединение основного класса с классом RootLayoutController

В некоторых местах кода класс `RootLayoutController` требует ссылку на класс `MainApp`. Пока мы эту ссылку еще не передали.

Откройте класс `MainApp` и замените метод `initRootLayout()` следующим кодом:

Обратите внимание на два изменения: на строки, *дающие доступ контроллеру к основному классу приложения* и на три последних строки для *загрузки последнего открытого файла с записями*.

Тестирование

Устроив тест-драйв вашему приложению убедитесь, что вы способны использовать меню для сохранения записей адресов в файл.

Когда вы откроете `xml`-файл в редакторе, то вместо значения дня рождения вы увидите пустой тег `<birthday/>`. Дело в том, что JAXB не знает как сконвертировать тип `LocalDate` в XML. Поэтому мы должны предоставить собственный класс `LocalDateAdapter` и определить процесс конвертации вручную.

Создайте новый класс `LocalDateAdapter` внутри пакета `ch.makery.address.util` и скопируйте туда следующий код:

LocalDateAdapter.java

Потом откройте класс `Person.java` и добавьте к методу `getBirthday()` следующую аннотацию:

Теперь протестируйте приложение еще раз. Попытайтесь сохранить и загрузить xml-файл с данными. После перезапуска, приложение должно автоматически загрузить последний открытый файл.

Как это работает

Давайте посмотрим как все это работает вместе:

1. Приложение запускается через метод `main(...)` из класса `MainApp`.
2. Вызывается конструктор `public MainApp()` и добавляет некоторые тестовые данные.

3. Дальше запускается метод `start(...)` в классе `MainApp` и вызывает метод `initRootLayout()` для инициализации корневой разметки с файла `RootLayout.fxml`. Файл fxml владеет информацией о том, какой контроллер использовать и связывает представление с `RootLayoutController`'ом.
4. Класс `MainApp` получает ссылку на `RootLayoutController` с fxml-загрузчика и передает этому контроллеру ссылку на себя. С этой ссылкой контроллер потом сможет иметь доступ к публичным методам класса `MainApp`.
5. В конце метода `initRootLayout` мы пытаемся вытянуть путь к последнему открытому файлу записей с настроек `Preferences`. Если настройки знают о таком файле, то мы загружаем с него данные. Эта процедура перезапишет наши тестовые данные, которые мы загружали в конструкторе.

Что дальше?

В 6 Части Учебника мы добавим статистический график дней рождений.

Вам могут быть интересны также некоторые другие статьи

- JavaFX Dialogs
- JavaFX Date Picker
- JavaFX Event Handling Examples
- JavaFX TableView Sorting and Filtering
- JavaFX TableView Cell Renderer

Screenshot AddressApp Part 5

Screenshot AddressApp Part 5

Часть 5: Содержание

- **Хранение данных в XML**
- Использование компонента JavaFX **FileChooser**
- Использование компонента JavaFX **Menu**
- Сохранение пути к последнему открытому файлу в пользовательских настройках

На данный момент наше приложение умеет хранить данные только в памяти. Каждый раз, когда мы закрываем приложение наши данные теряются. Поэтому настало время подумать о постоянном хранении данных.

Сохранение пользовательских настроек

Java позволяет нам хранить некоторые данные о состоянии приложения в классе `Preferences`. В зависимости от операционной системы, класс `Preferences` записывает данные в разные места (например, в файл регистра в ОС Windows).

Мы не можем использовать класс `Preferences` для хранения всей нашей записной книжки. Но мы можем сохранить там некоторые настройки приложения, например, путь к последнему открытому файлу. Имея эти данные, мы сможем восстанавливать состояние нашего приложения после перезагрузки.

Следующие два метода обеспечивают сохранение и восстановление настроек нашего приложения. Добавьте их в конец класса `MainApp`:

MainApp.java

Хранение данных в XML

Почему именно XML?

Использование баз данных является одним из наиболее распространенных способов хранения данных. Пока наши данные, которые мы должны хранить, являются объектами, базы данных содержат их в виде реляционных данных (например, таблиц). Это называется object-relational impedance mismatch. Но для того, чтобы привести наши объектные данные в соответствие с реалицяонными таблицами требуется выполнить дополнительную работу. Существуют фреймворки, которые помогают приводить наши объектные данные в соответствие с реляционной базой данных (Hibernate - один из наиболее популярных), но чтобы начать их использовать, также необходимо проделать дополнительную работу на настройку.

Для нашего простого приложения намного легче хранить данные в виде XML. Для этого мы будем использовать библиотеку JAXB (**J**ava **A**rchitechture for **X**ML **B**inding). Написав всего несколько строк кода, JAXB позволит нам сгенерировать такой исходный XML-файл:

Пример сгенерированного XML-файла

Использование JAXB

Библиотека JAXB изначально включена в JDK. Это значит, что никаких дополнительных библиотек нам подключать не придется.

JAXB предоставляет две основные функции: способность к **маршаллированию** Java-объектов в XML и обратную **демаршализацию** с xml-файла в Java-объекты.

Для того, чтобы JAXB предоставила возможность такой конвертации, нам необходимо подготовить нашу модель.

Подготовка класса-модели для JAXB

Данные, которые мы хотим сохранить находятся в переменной `personData` в классе `MainApp`. JAXB требует пометить аннотацией `@XmlRootElement` внешний класс наших данных (только класс, поле этой аннотацией пометить нельзя). Тип переменной `personData` является `ObservableList`, на который мы не можем поставить аннотацию. Поэтому, для того, чтобы решить эту проблему нам необходимо создать класс-обертку, который будет содержать только нашу коллекцию записей, и на который мы поставим аннотацию `@XmlRootElement`.

Создайте в пакете `model` новый класс `PersonListWrapper`.

PersonListWrapper.java

Обратите внимание на две аннотации:

- `@XmlRootElement` определяет имя корневого элемента.
- `@XmlElement` оппределяет имя элемента, которое нам указывать необязательно.

Чтение и запись данных с помощью JAXB

Сделаем наш класс `MainApp` ответственным за чтение и запись данных нашего приложения. Для этого добавьте два метода в конец класса `MainApp.java`:

Возможность маршаллинга и демаршализации готовы. Теперь давайте создадим пункты меню "*Сохранить*" и "*Загрузить*" для того, чтобы использовать данные возможности.

Обработка действий меню

Мы уже создавали меню в файле `RootLayout.fxml`, которое пока еще не использовали. Перед тем, как мы добавим поведение нашему меню, давайте создадим все необходимые нам пункты.

Откройте файл `RootLayout.fxml` в приложении Scene Builder и перенесите необходимое количество пунктов меню из вкладки *Library* на вкладку *Hierarchy*. Создайте следующие пункты меню: **New**, **Open...**, **Save**, **Save as...** и **Exit**.

Add Menu Items

Add Menu Items

Подсказка: Используйте свойство *Accelerator* в вкладке *Properties* для установки горячих клавиш на пункты меню.

Класс RootLayoutController

Для обработки поведения меню нам необходимо новый класс-контроллер. Создайте класс `RootLayoutController` в пакете `ch.makery.address.view`.

Добавьте к классу-контроллеру следующее содержание:

RootLayoutController.java

Компонент FileChooser

Обратите внимание на методы, которые используют компонент `FileChooser` в классе `RootLayoutController`. Сперва мы создаем новый экземпляр класса `FileChooser`. Потом мы применяем фильтр расширешия и при выборе файлов будут показываться только те, которые имеют расширение `.xml`. Ну и наконец, мы отображаем данный компонент поверх нашей *PrimaryStage*.

Если пользователь закрывает диалог выбора файлов не выбрав ни никакого файла, тогда возвращается `null`. В противном случае мы берем выбранный файл и передаем его внутрь метода `loadPersonDataFromFile(...)` или `savePersonDataToFile(...)`, которые находятся в классе `MainApp`.

Соединение fxml-файла с классом-контроллером

1. Откройте файл `RootLayout.fxml` в приложении Scene Builder. На вкладке *Controller* в качестве класса-контроллера выберите значение `RootLayoutController`.

2. Перейдите на вкладку *Hierarchy* и выберите пункт меню. Во вкладке *Code* в качестве значений свойства *On Action* вы сможете увидеть все доступные методы выбранного класса-контроллера. Выберите метод, соответствующий данному пункту меню.
   Menu Actions

3. Повторите предыдущий шаг для каждого пункта меню.

4. Закройте приложение Scene Builder и оновите ваш проект (нажмите **Refresh (F5)** на корневой папке вашего проекта). Это позволит среде разработки Eclipse увидеть сделанные вами изменения в приложении Scene Builder.

Соединение основного класса с классом RootLayoutController

В некоторых местах кода класс `RootLayoutController` требует ссылку на класс `MainApp`. Пока мы эту ссылку еще не передали.

Откройте класс `MainApp` и замените метод `initRootLayout()` следующим кодом:

Обратите внимание на два изменения: на строки, *дающие доступ контроллеру к основному классу приложения* и на три последних строки для *загрузки последнего открытого файла с записями*.

Тестирование

Устроив тест-драйв вашему приложению убедитесь, что вы способны использовать меню для сохранения записей адресов в файл.

Когда вы откроете `xml`-файл в редакторе, то вместо значения дня рождения вы увидите пустой тег `<birthday/>`. Дело в том, что JAXB не знает как сконвертировать тип `LocalDate` в XML. Поэтому мы должны предоставить собственный класс `LocalDateAdapter` и определить процесс конвертации вручную.

Создайте новый класс `LocalDateAdapter` внутри пакета `ch.makery.address.util` и скопируйте туда следующий код:

LocalDateAdapter.java

Потом откройте класс `Person.java` и добавьте к методу `getBirthday()` следующую аннотацию:

Теперь протестируйте приложение еще раз. Попытайтесь сохранить и загрузить xml-файл с данными. После перезапуска, приложение должно автоматически загрузить последний открытый файл.

Как это работает

Давайте посмотрим как все это работает вместе:

1. Приложение запускается через метод `main(...)` из класса `MainApp`.
2. Вызывается конструктор `public MainApp()` и добавляет некоторые тестовые данные.
3. Дальше запускается метод `start(...)` в классе `MainApp` и вызывает метод `initRootLayout()` для инициализации корневой разметки с файла `RootLayout.fxml`. Файл fxml владеет информацией о том, какой контроллер использовать и связывает представление с `RootLayoutController`'ом.
4. Класс `MainApp` получает ссылку на `RootLayoutController` с fxml-загрузчика и передает этому контроллеру ссылку на себя. С этой ссылкой контроллер потом сможет иметь доступ к публичным методам класса `MainApp`.
5. В конце метода `initRootLayout` мы пытаемся вытянуть путь к последнему открытому файлу записей с настроек `Preferences`. Если настройки знают о таком файле, то мы загружаем с него данные. Эта процедура перезапишет наши тестовые данные, которые мы загружали в конструкторе.

Что дальше?

В 6 Части Учебника мы добавим статистический график дней рождений.

Вам могут быть интересны также некоторые другие статьи

- JavaFX Dialogs
- JavaFX Date Picker
- JavaFX Event Handling Examples
- JavaFX TableView Sorting and Filtering
- JavaFX TableView Cell Renderer

Screenshot AddressApp Part 7

Screenshot AddressApp Part 7

Я думаю, что написал последнюю часть данного учебника чтобы показать вам как развернуть (т.е. упаковать и опубликовать) наше приложение.

Часть 7: Содержание

- Развертывание нашего JavaFX-приложения в виде нативной упаковки (Native Package) с использованием плагина e(fx)clipse.

Что такое развертывание

Развертывание - это процесс упаковки и доставки програмного обеспечения к пользователю. Это важная часть процесса разработки ПО, т.к. здесь происходит первый контакт пользователя с нашим приложением.

Java рекламируется под слоганом *“Написано однажды, работает везде”* для того, чтобы проиллюстрировать *кроссплатформенное* преимущество данного языка программирования. В идеале, это означает, что наше java-приложение сможет запуститься на любом устройстве, где установлена виртуальная машина Java (JVM).

В прошлом, пользовательский опыт при установке java-приложения не всегда был гладким. Если у пользователя на системе не была установлена требуемая версия Java, он перенаправлялся для ее инсталляции. Это приводило к некоторым трудностям: к необходимости иметь права администратора, к возникновению вопросов совместимости версий Java, и т.д.

К счастью, JavaFX предоставляет новую опцию развертывания, которая имеет название “Нативная упаковка” (**Native Packaging**) (другое название “Автономная упаковка приложения” (Self-Containde Application Package)). Нативная упаковка - это пакет, содержащий вместе java-приложение и среду выполнения Java для конкретной платформы.

Официальная документация по JavaFX от Oracle содержит обширное руководство для всех возможных вариантов развертывания JavaFX-приложений.

В этой статье я покажу как создать “Нативную упаковку” с помощью приложения Eclipse и плагина e(fx)clipse.

Создание нативной упаковки

Цель заключается в создании автономно-упакованного приложения, размещенного в одной директории на компьютере пользователя. То, как оно примерно будет выглядеть для нашего приложения на Windows можно посмотреть ниже:

AddressApp Native Package

AddressApp Native Package

Папка `app` содержит данные нашего приложения, а папка `runtime` - платформо-зависимую среду выполнения.

Для того, чтобы сделать этот процесс более комфортным для пользователя, мы предоставим ему инсталлятор:

- `exe`-инсталлятор для Windows
- `dmg`-инсталлятор для MacOS.

Плагин e(fx)clipse поможет нам сгенерировать нативную упаковку и инсталлятор.

Шаг 1 - Редактируем файл build.fxbuild

Файл `build.fxbuild` используется плагином e(fx)clipse для генерации файла, который будет использоватся инструментом сборки Ant. (Если у вас нет файла `build.fxbuild`, создайте новый JavaFX-проект в приложении Eclipse и скопируйте сгенерированный файл.)

1. Откройте файл `build.fxbuild` из корневой папки вашего проекта.
2. Заполните все поля помеченные звездочкой. Для MacOS: не используйте пробелы в названии приложения (в поле *Application Title*) так как это, кажется, приводит к проблемам).
   fxbuild settings
3. Выберите значение формата упаковки: для Windows - `exe`, для MacOS - `dmg`, для Linux - `rpm`.
4. Нажмите на ссылку `Generate ant build.xml only` справа.
   generate ant build
5. Проверьте, создалась ни новая папка `build` с файлом `build.xml`.

Шаг 2 - Добавляем в инсталлятор иконки

Мы можем добавить в наш инсталлятор пару красивых иконок:

- AddressApp.ico для иконки инсталляционного файла
- AddressApp-setup-icon.bmp для экрана приветствия инсталлятора
- AddressApp.icns - для инсталлятора под MacOS

1. Создайте следующие подпапки в папке `build`:
- `build/package/windows` (используется только в Windows)
- `build/package/macos` (используется только в MacOS)

2. Скопируйте соответствующие иконки с приведенных ссылок и поместите их в назначенные для них папки. В результате это должно выглядеть так:
   Installer Icons
3. **Важно**: Имена иконок должны точно совпадать с именем приложения, которое вы указали в `build.fxbuild`:
   - `YourAppTitle.ico`
   - `YourAppTitle-setup-icon.bmp`
   - `YourAppTitle.icns`

Шаг 3 - Добавляем ресурсы

Наша папка `resources` не скопируется автоматически. Мы должны в ручную добавить ее в папку `build`:

1. Создайте подпапку `dist` в папке `build`:
   - `build/dist`

2. Скопируйте папку `resources` (которая содержит иконки для нашего приложения) в `build/dist`.
   Build Resources

Шаг 4 - Редактируем build.xml и включаем в него иконки

Плагин e(fx)clipse сгенерировал файл `build/build.xml`, который готов для выполнения **Ant**'ом. Пока наши иконки работать не будут.

Так как e(fx)clipse не может сообщить (пока?), что нам необходимо включить дополнительные ресурсы в папку, нам необходимо сделать это вручную путем редактирования `build.xml`:

Откройте `build.xml` и найдите тег `<path id="fxant">`. Добавьте одну строку для `${basedir}` (это сделает наши иконки видимыми):

build.xml - добавляем "basedir"

Ниже найдите блок кода `fx:resources id="appRes"`. Добавьте одну строку для наших ресурсов:

build.xml - добавляем "resources"

Почему-то, номер версии не добавляется в `fx:application`, который всегда делает версию нашего установщик `1.0` (как указали несколько человек в комментариях). Для исправления этого, добавьте номер версии вручную (спасибо Марку, что выяснил это):

build.xml - добавляем "version"

Теперь мы можем запускать `build.xml` через Ant. Это сгенерирует нам работоспособный jar-файл проекта. Но мы хотим пойти на шаг дальше и создать хороший установщик.

Шаг 5 - exe-установщик под Windows

AddressApp on Windows

AddressApp on Windows

С помощью **Inno Setup** мы можем создать установщик под Windows для нашего приложения в виде единого `.exe`-файла. Созданный `.exe`-установщик будет проводить установку программы на уровне пользователя (не потребуются права администратора). Также будет создан ярлык (в меню или на рабочем столе).

1. Скачайте Inno Setup версии 5 или выше. Установите Inno Setup на ваш компьютер. Наш Ant-скрипт будет использоваться для автоматической генерации установщика.

2. Укажите Windows путь к Inno Setup (например, `C:\Program Files (x86)\Inno Setup 5`). Для этого добавьте Inno Setup к переменной `Path` в ваших переменных окружения. Если вы не знаете где их найти, почитайте Как установить пути и переменные окружения в Windows.

3. Перезапустите приложение Eclipse и продолжите с шага 5.

Шаг 5 - dmg-установщик для MacOS

AddressApp on Mac

AddressApp on Mac

Для создания `dmg`-установщика под MacOS никаких дополнительных инструментов не требуется.

Заметка: Для того, чтобы отображалось изображение установщика, вы должны задать этому изображению такое же название, как у вашего приложения.

Шаг 5 - rpm-установщик для Linux

Для других способов упаковки приложения (`msi` для Windows, `rpm` для Linux) читайте эту статью или документацию от Oracle.

Шаг 6 - Запускаем build.xml

На последнем шаге мы запустим файл `build.xml` с помощью Ant: кликните правой кнопкой мышки на файл `build.xml` | *Run As* | *Ant Build*.

Run Ant Build

Run Ant Build

Сборка приложения займет немного времени (около 1 минуты на моем компьютере).

Если все прошло удачно, вы должны найти нативные сборки в папке `build/deploy/bundles`. Вот так выглядит версия для Windows:

Deployed File

Deployed File

Файл `AddressApp-1.0.exe` может использоваться как единственный файл для установки приложения. Этот установщик скопирует нашу сборку в папку `C:/Users/[yourname]/AppData/Local/AddressApp`.

Что дальше?

Надеюсь, что этот учебник поможет вам начать работать с JavaFX и с этого момента вы сможете писать свои JavaFX-проекты.

Я ценю любую обратную связь. Не стесняйтесь писать комментарии если у вас есть какие-либо предложения или если вам что-то не ясно.

Вам могут быть интересны также некоторые другие статьи

- JavaFX Dialogs
- JavaFX Date Picker
- JavaFX Event Handling Examples
- JavaFX TableView Sorting and Filtering
- JavaFX TableView Cell Renderer