Tutorial JavaFX

Libreta de direcciones con JavaFX

Enlace del proyecto original

Mayo 2020

Índice general

Introducción

Por que el proyecto

El tutorial original fue escrito en inglés y traducido al español por Mario Gómez Martínez, en este documento la intención es realizar el mismo proyecto con las actulizaciones a la última versión del JDK, la codificación se actualizará al español con los respectivos comentarios.

Introducción

JavaFX proporciona a los desarrolladores de Java una nueva plataforma gráfica. JavaFX 2.0 se publicó en octubre del 2011 con la intención de reemplazar a Swing en la creación de nuevos interfaces gráficos de usuario (IGU). Cuando empecé a enseñar JavaFX en 2011 era una tecnología muy incipiente todavía. No había libros sobre JavaFX que fueran adecuados para estudiantes de programación noveles, así es que empecé a escribir una serie de tutoriales muy detallados sobre JavaFX.
El tutorial te guía a lo largo del diseño, programación y publicación de una aplicación de contactos (libreta de direcciones) mediante JavaFX. Este es el aspecto que tendrá la aplicación final:
Colocar la imagen después

Lo que aprenderás

- Creación de un nuevo projecto JavaFX
- Uso de Scene Builder para diseñar la interfaz de usuario
- Estructuración de una aplicación según el patrón MVC (Modelo, Vista, Controlador)
- Uso de `ObservableList` para la actualización automática de la interfaz de usuario
- Uso de `TableView` y respuesta a cambios de selección en la tabla
- Creación de un diálogo personalizado para editar personas
- Validación de la entrada del usuario

- Aplicación de estilos usando CSS
- Persistencia de datos mediante XML
- Guardado del último archivo abierto en las preferencias de usuario
- Creación de un gráfico JavaFX para mostrar estadísticas
- Despliegue de una aplicación JavaFX nativa

Después de completar esta serie de tutoriales deberías estar preparado para desarrollar aplicaciones sofisticadas con JavaFX.

Cómo usar este tutorial

Hay dos formas de utilizarlo

- **Máximo-aprendizaje:** Crea tu propio proyecto JavaFX desde cero.
- **Máxima-rápidez:** Importa el código fuente de una parte del tutorial en tu entorno de desarrollo favorito (es un proyecto Eclipse, pero puedes usar otros entornos, como Netbeans, con ligeras modificaciones). Después revisa el tutorial para entender el código. Este enfoque también resulta útil si te quedas atascado en la creación de tu propio código.

Parte 1

Eclipse y Scene Builder

1.1. Contenidos en Parte 1

- Familiarizándose con JavaFX.
- Crear y empezar un proyecto JavaFX.
- Uso de Scene builder para diseñar la interfaz de usuario
- Estructura básica de una aplicación mediante el patrón Modelo Vista Controlador (MVC).

1.2. Prerrequisitos

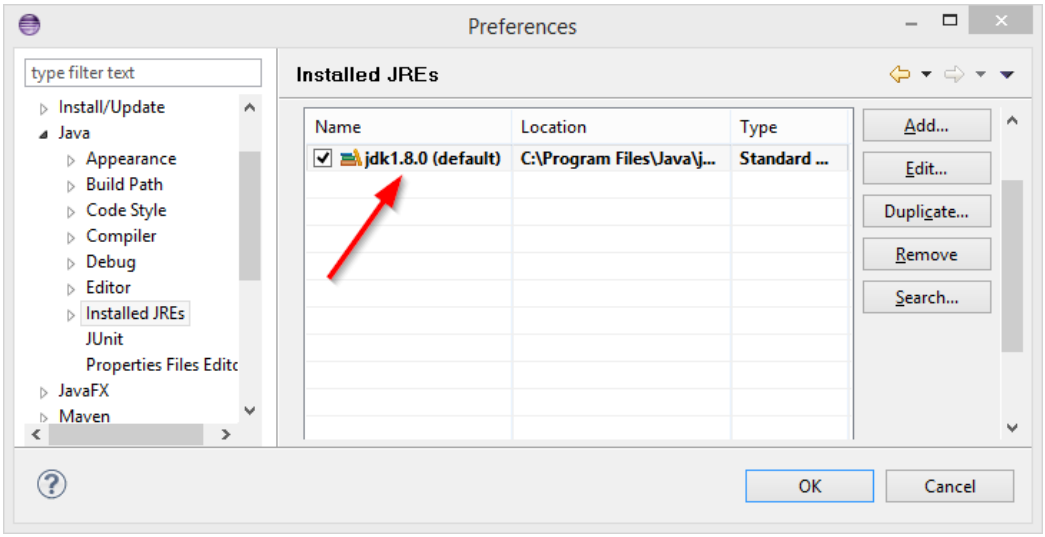
- Última versión de Java JDK 8 o superior.
- Eclipse 4.3 o superior con el plugin e(fx)clipse. La forma más sencilla de obtenerlo es descargarse la distribución preconfigurada desde e(fx)clipse website. Como alternativa puedes usar un sitio de actualización para tu instalación de Eclipse.
- Scene Builder 2.0 o superior

1.3. Configuración de Eclipse

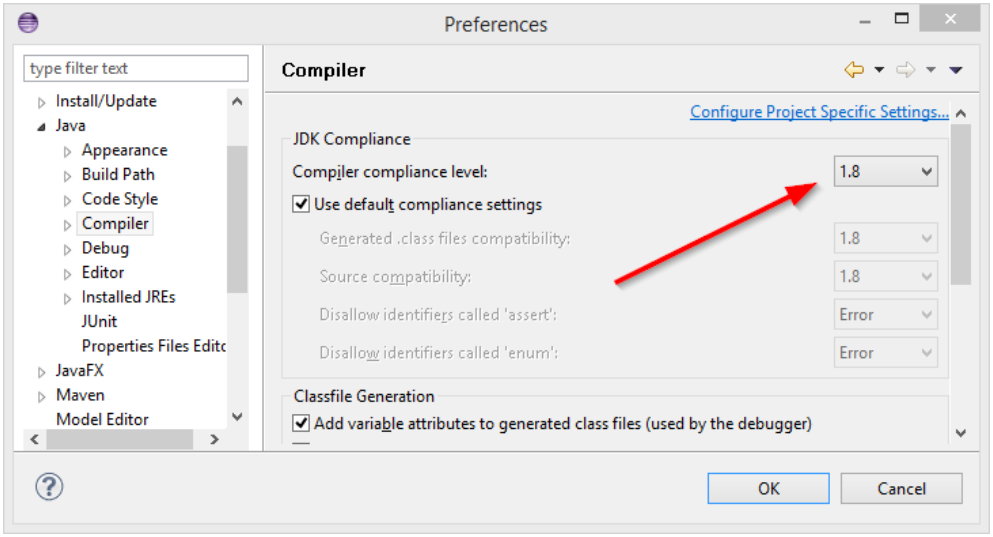
Hay que indicarle a Eclipse que use JDK 8 y también dónde se encuentra el ejecutable del Scene Builder:

1. Abre las Preferencias de Eclipse (menú *Window — Preferences* y navega hasta *Java — Installed JREs*.
2. Si no lo tienes el jre1.8 en la lista de JREs, entonces pulsa *Add...*, selecciona *Standard VM* y elige el Directorio de instalación (JRE Home directory) de tu JDK 8.

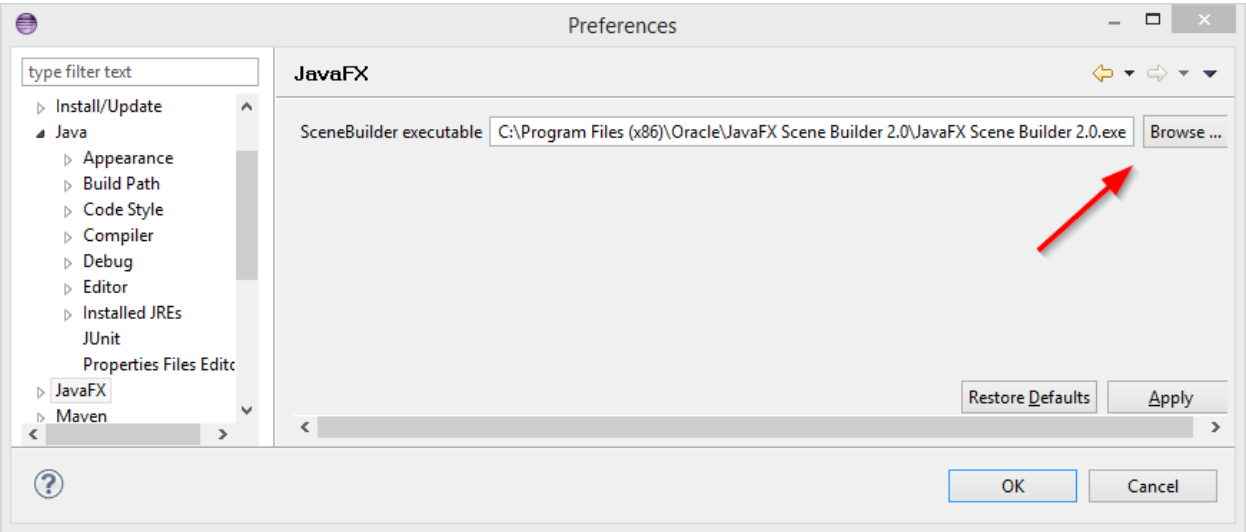
3. Elimina otros JREs o JDKs de tal manera que **JDK 8 se convierta en la opción por defecto**.



4. Navega a *Java* — *Compiler*. Establece el **nivel de cumplimiento del compilador en 1.8** (Compiler compliance level).



5. Navega hasta *Java* — *JavaFX*. Especifica la ruta al ejecutable del Scene Builder.



1.4. Enlaces útiles

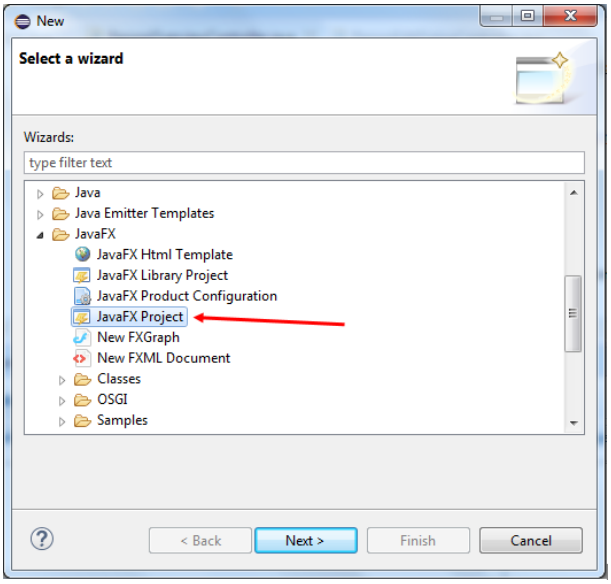
Te podría interesar los siguientes enlaces:

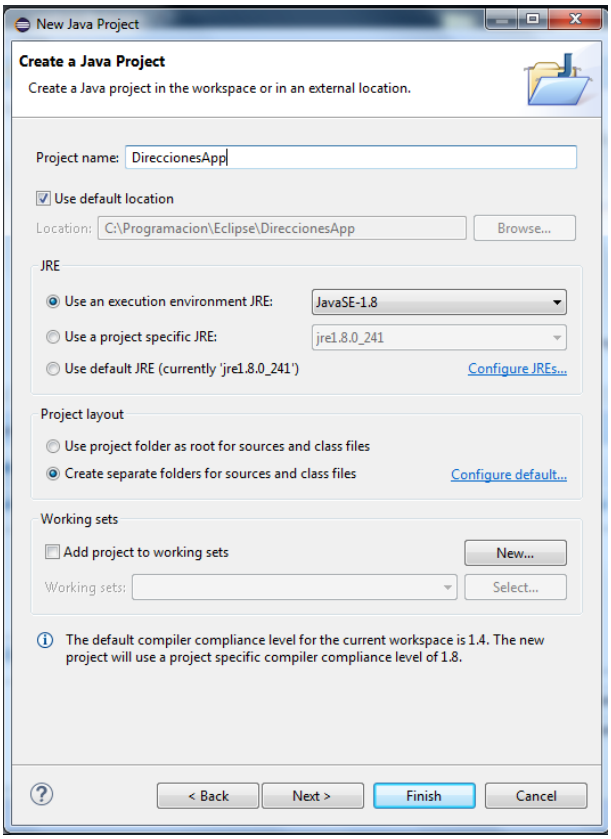
1. Java 8 API - Documentación (JavaDoc) de las clases estándar de Java
2. JavaFX 8 API - Documentación de las clases JavaFX
3. ControlsFX API - Documentación (JavaDoc) - Documentación para el proyecto ControlsFX, el cual ofrece controles JavaFX adicionales
4. Oracle's JavaFX Tutorials - Tutoriales oficiales de Oracle sobre JavaFX

`¡Y ahora, manos a la obra!`

1.5. Crea un nuevo proyecto JavaFX

En Eclipse (con e(fx)clipse instalado) ve a *File — New — Other...* y elige *JavaFX Project*. Especifica el nombre del proyecto (ej. DireccionesApp) y aprieta *Finish*.





Borra el paquete *application* y su contenido que ha sido creado automáticamente.

1.5.1. Crea los paquetes

Desde el principio vamos a seguir buenos principios de diseño de software. Algunos de estos principios se traducen en el uso de la arquitectura denominada Modelo-Vista-Controlador

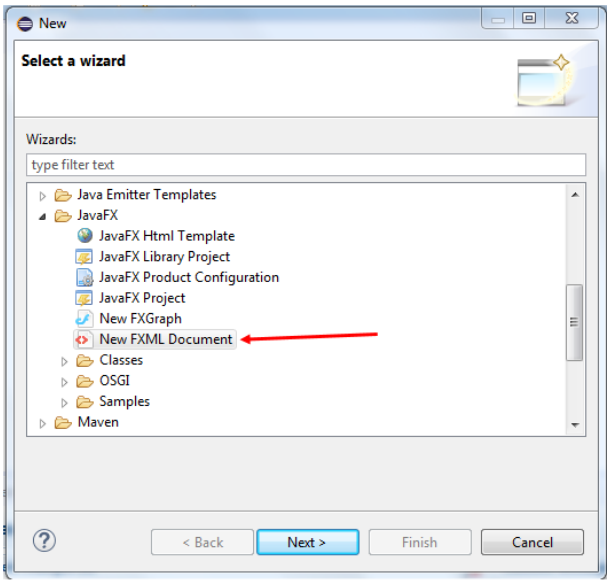
(MVC). Esta arquitectura promueve la división de nuestro código en tres apartados claramente definidos, uno por cada elemento de la arquitectura. En Java esta separación se logra mediante la creación de tres paquetes separados.
En el ratón hacemos clic derecho en la carpeta *src, New — Package:*

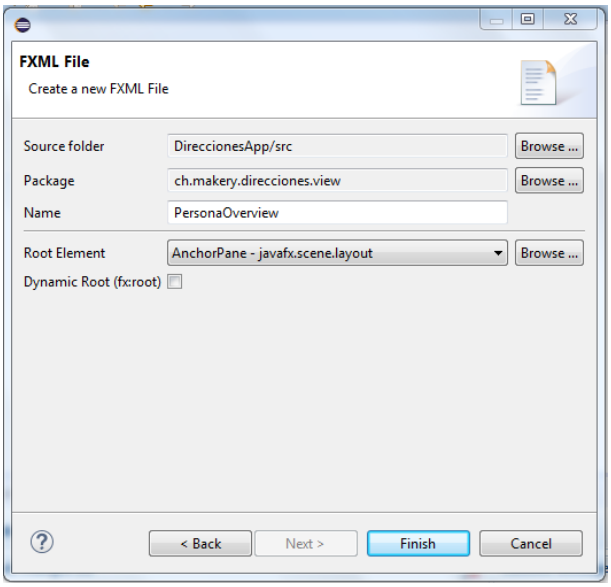
- `ch.makery.direcciones` - contendrá la mayoría de clases de control (C).
- `ch.makery.direcciones.model` - contendrá las clases del modelo (M).
- `ch.makery.direcciones.view` - contendrá las vistas (V)

Nota: Nuestro paquete dedicado a las vistas contendrá también algunos controladores dedicados exclusivamente a una vista. Les llamaremos **controladores-vista**.

1.6. Crea el archivo FXML de diseño

Hay dos formas de crear la interfaz de usuario. Programándolo en Java o mediante un archivo XML. Si buscas en Internet encontrarás información relativa a ambos métodos. Aquí usaremos XML (archivo con la extensión .fxml) para casi todo. Encuentro más claro mantener el controlador y la vista separados entre sí. Además, podemos usar la herramienta de edición visual Scene Builder, la cual nos evita tener que trabajar directamente con el XML.
Haz clic derecho el paquete *view* y crea un nuevo archivo FXML (*New — Other — FXML — New FXML Document*) llamado `PersonaOverview`.

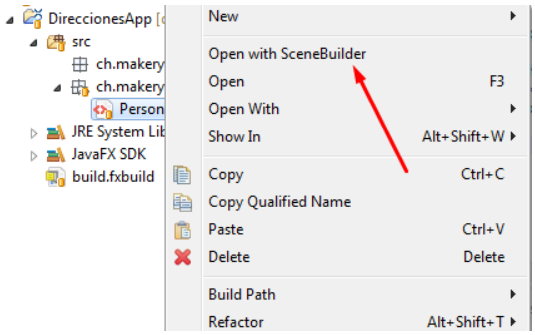




1.7. Diseño mediante Scene Builder

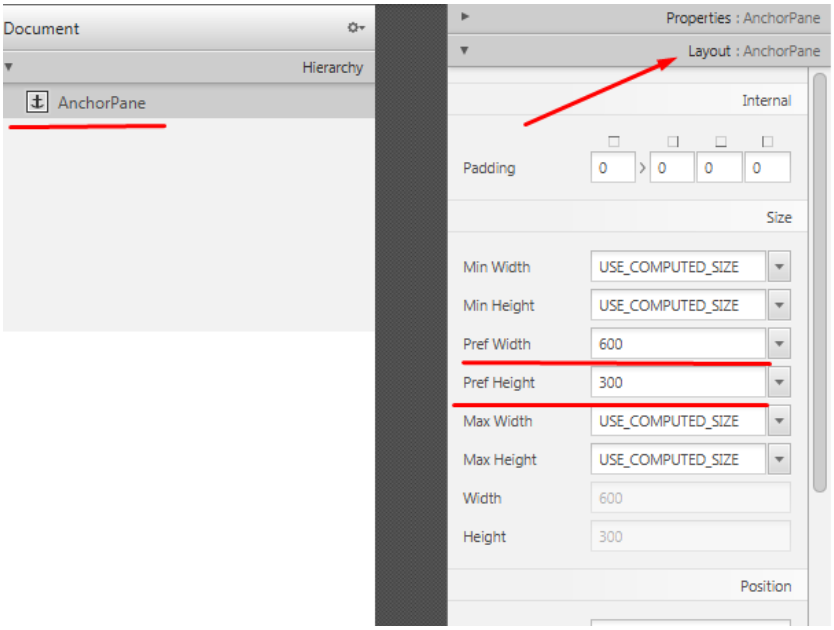
> **Nota:** Si no puedes hacerlo funcionar, descarga las fuentes para esta parte del tutorial e inténtalo con el archivo fxml incluido.

Haz clic derecho sobre PersonaOverview.fxml y elige *Open with Scene Builder*. Ahora deberías ver el Scene Builder con un AnchorPane (visible en la jerarquía de componentes (herramienta Hierarchy) situada a la izquierda).

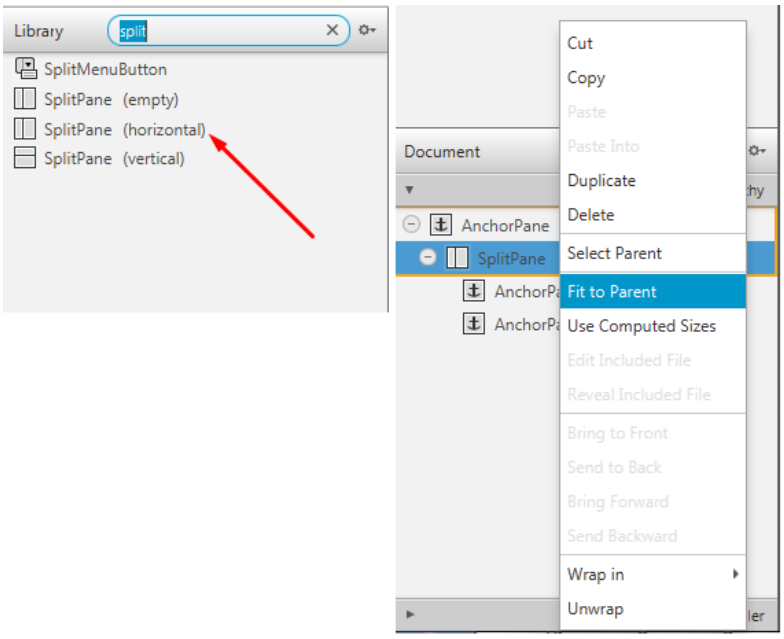


Ya abierto Scene Builder

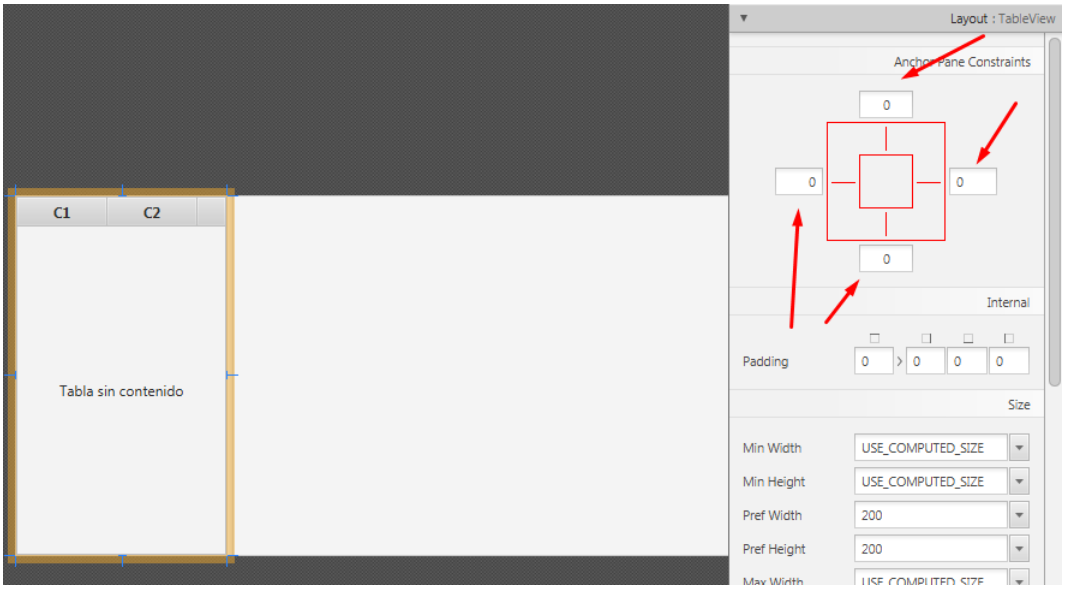
1. Selecciona el *AnchorPane* en tu jerarquía y ajusta el tamaño en el apartado Layout (a la derecha):



2. Añade un *SplitPane (Horizontal Flow)* arrastrándolo desde la librería (Library) al área principal de edición. Haz clic derecho sobre el SplitPane en la jerarquía y elige Fit to Parent.

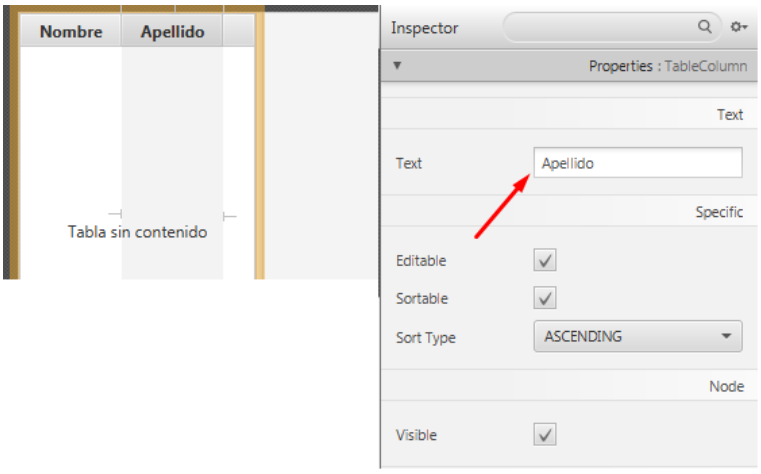


3. Arrastra un TableView (bajo Controls) al lado izquierdo del SplitPane. Selecciona la TableView (no una sola columna) y establece las siguientes restricciones de apariencia (Layout) para la TableView. Dentro de un AnchorPane siempre se pueden establecer anclajes (anchors) para los cuatro bordes (más información sobre Layouts).

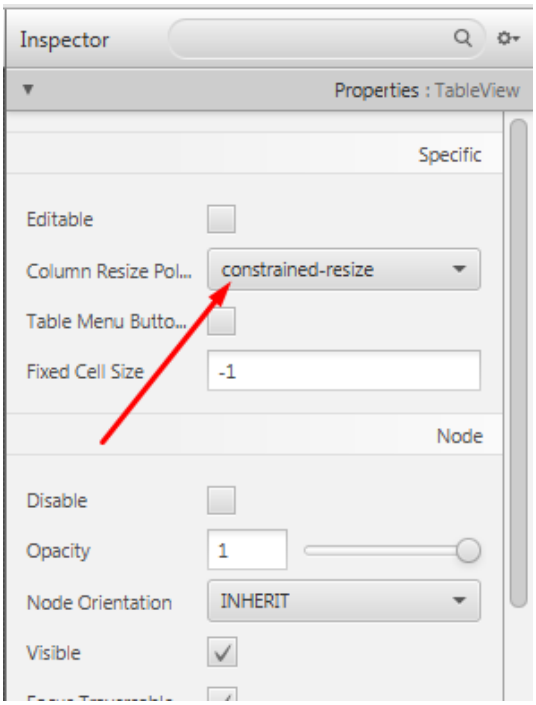


4. Ve al menú *Preview — Show Preview in Window* para comprobar si se visualiza correctamente. Intenta cambiar el tamaño de la ventana. La TableView debería ajustar su tamaño al tamaño de la ventana, pues está "anclada" a sus bordes.

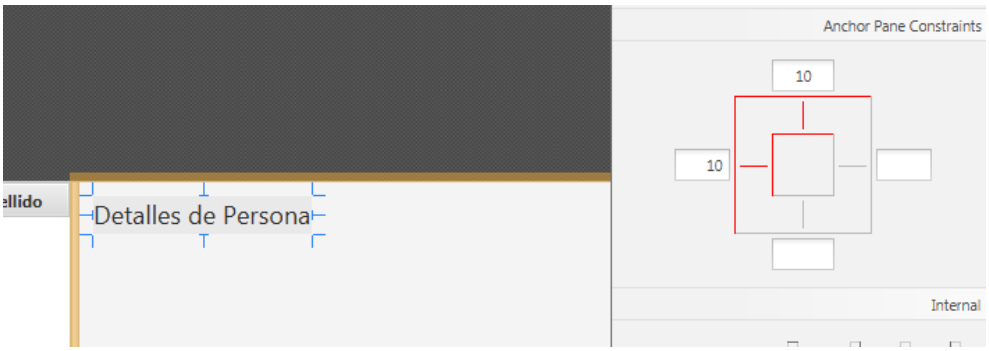
5. Cambia el texto de las columnas (en *Properties*) a "Nombre" y "Apellido".



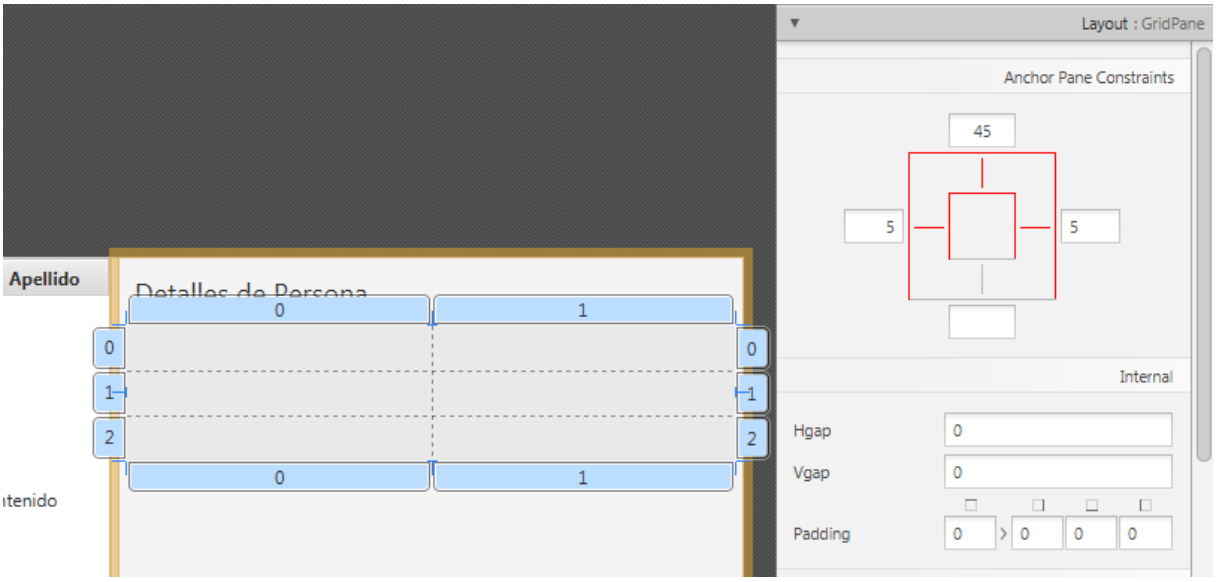
6. Selecciona la *TableView* y elige *constrained-resize* para la *Column Resize Policy* (en *Properties*). Esto asegura que las columnas usarán siempre todo el espacio que tengan disponible.



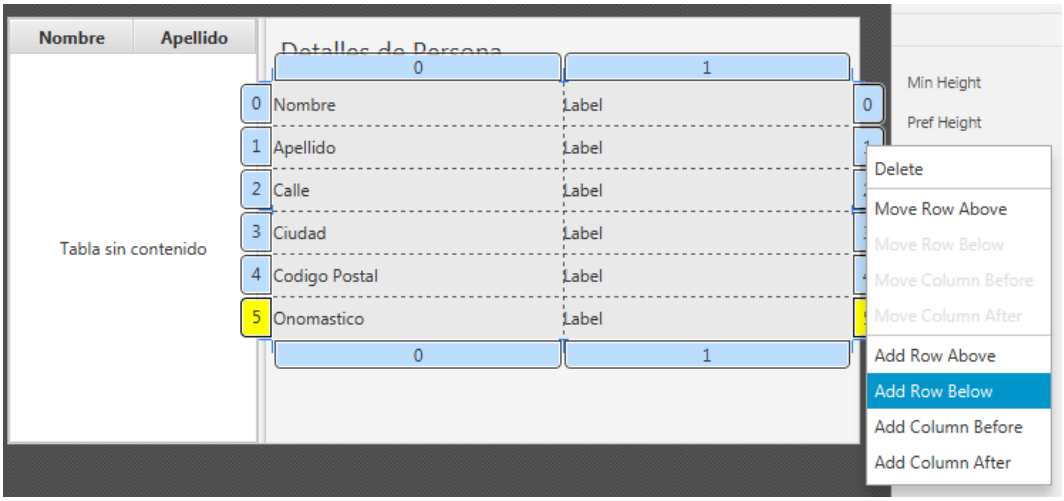
7. Añade una *Label* al lado derecho del *SplitPane* con el texto "Detalles de Persona" (truco: usa la búsqueda en la librería para encontrar el componente *Label*). Ajusta su apariencia usando anclajes.



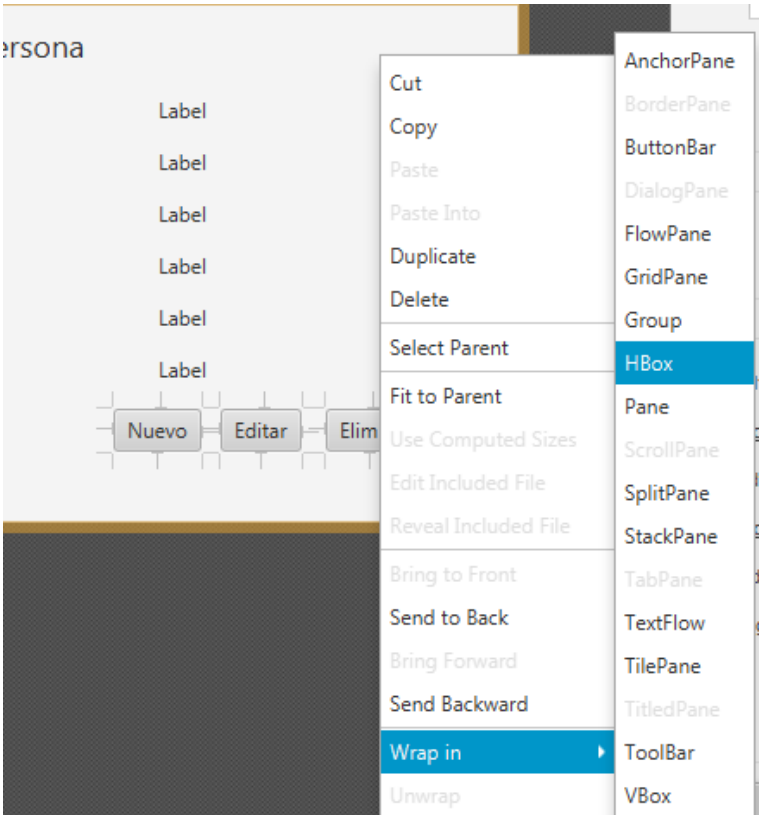
8. Añade un *GridPane* al lado derecho, selecciónalo y ajusta su apariencia usando anclajes (superior, derecho e izquierdo).

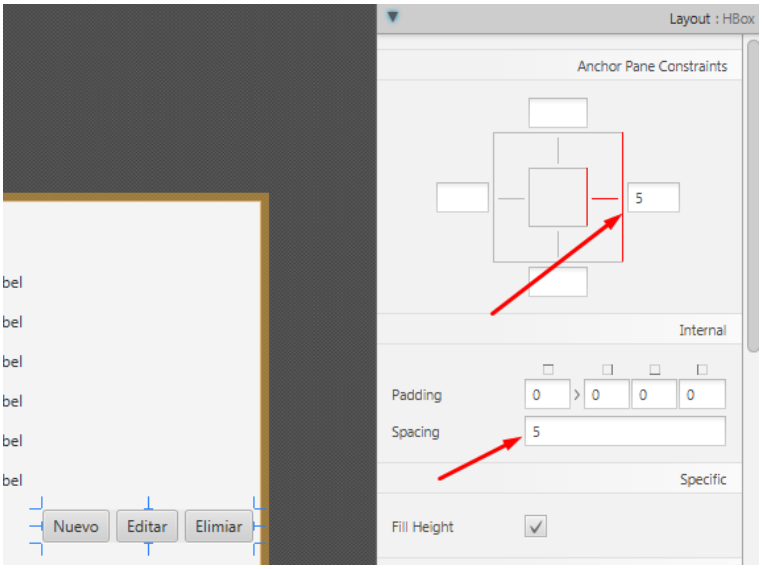


9. Añade las siguientes etiquetas (*Label*) a las celdas del GridPane.
   Nota: Para añadir una fila al *GridPane* selecciona un número de fila existente (se volverá de color amarillo), haz clic derecho sobre el número de fila y elige "Add Row".

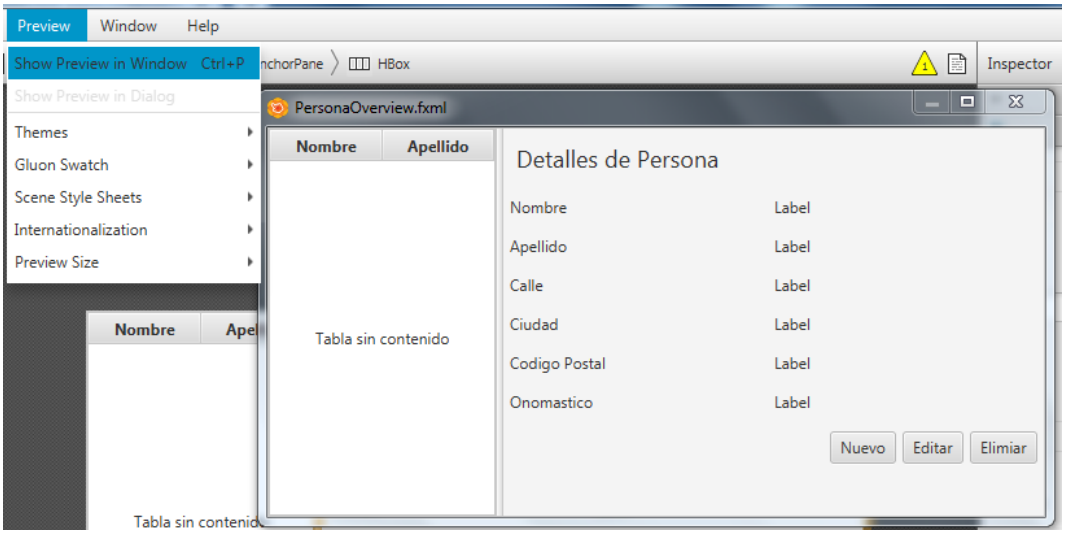


10. Añade tres botones a la parte inferior. Truco: Selecciónalos todos, haz click derecho e invoca *Wrap In — HBox*. Esto los pondrá a los 3 juntos en un *HBox*. Podrías necesitar establecer un espaciado spacing dentro del HBox. Después, establece también anclajes (derecho e inferior) para que se mantengan en el lugar correcto.





11. Ahora deberías ver algo parecido a lo siguiente. Usa el menú *Preview* para comprobar su comportamiento al cambiar el tamaño de la ventana.

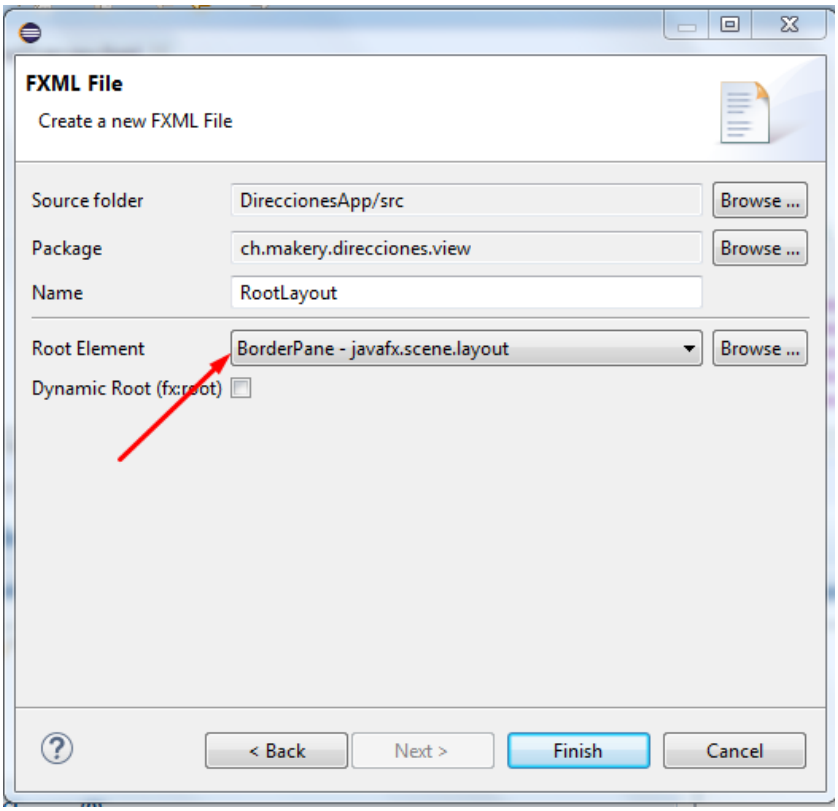


> **Nota:** Guarda los cambios en SceneBuilder, el archivo se actualizará automáticamente en Eclipse, si esto no sucede, puedes presionar F5 para actualizar el archivo.

1.8. Crea la aplicación principal

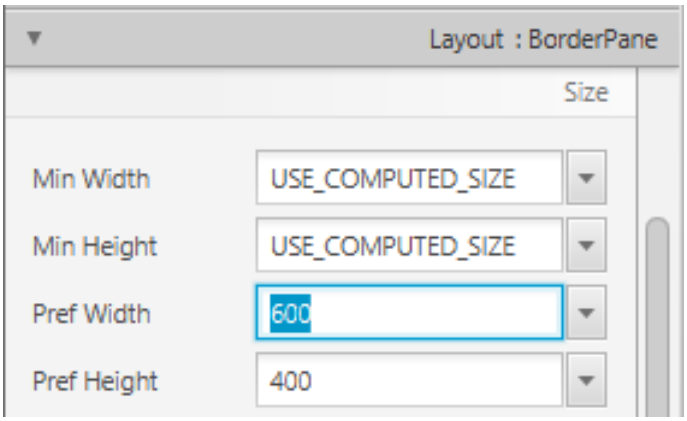
Necesitamos otro archivo FXML para nuestra vista raíz, la cual contendrá una barra de menús y encapsulará la vista recién creada `PersonaOverview.fxml`.

1. Crea otro archivo FXML dentro del paquete view llamado `RootLayout.fxml`. Esta vez, elige BorderPane como elemento raíz.



2. Abre `RootLayout.fxml` en el Scene Builder.

3. Cambia el tamaño del *BorderPane* con la propiedad *Pref Width* establecida en 600 y *Pref Height* en 400.

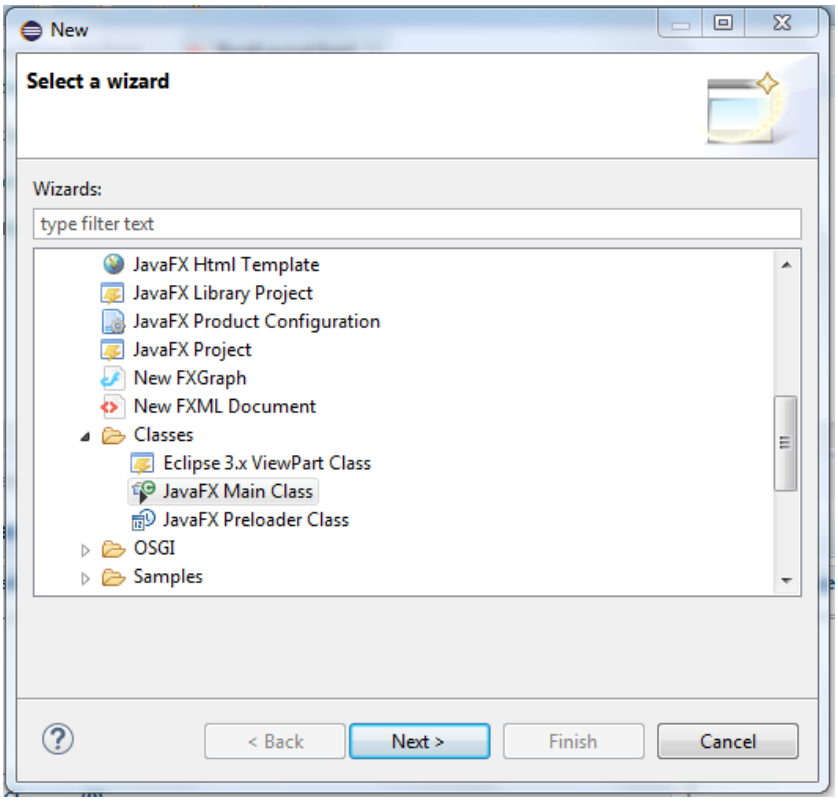


4. Añade una *MenuBar* en la ranura superior del *BorderPane*. De momento no vamos a implementar la funcionalidad del menú.

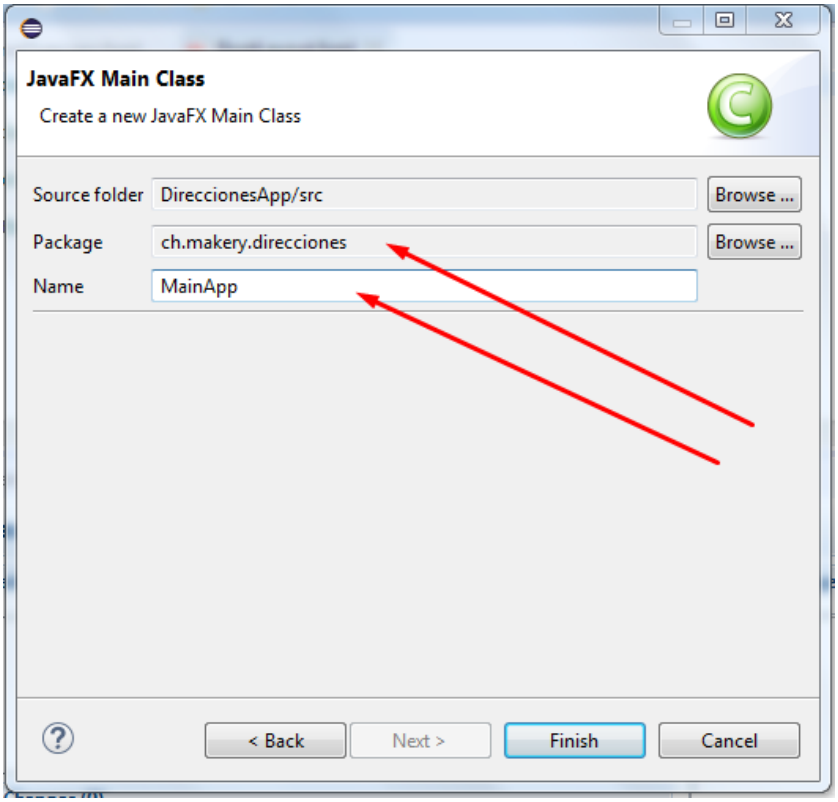
1.9. La clase principal en JavaFX

Ahora necesitamos crear la clase java principal, la cual iniciará nuestra aplicación mediante `RootLayout.fxml` y añadirá la vista `PersonOverview.fxml` en el centro.

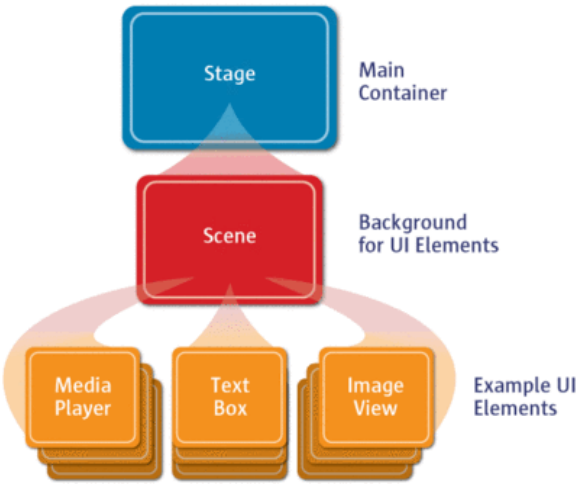
1. Haz clic derecho en el proyecto y elige *New* — *Other* — *JavaFX* — *classes* — *JavaFX Main Class*.



2. Llama a esta clase `MainApp` y ponla en el paquete de controladores `ch.makery.address` (nota: este es el paquete padre de los paquetes `view` y `model`).



La clase generada (`MainApp.java`) extiende a la clase `Application` y contiene dos métodos. Esta es la estructura básica que necesitamos para ejecutar una Aplicación JavaFX. La parte más importante para nosotros es el método `start(Stage primaryStage)`. Este método es invocado automáticamente cuando la aplicación es lanzada desde el método `main`. Como puedes ver, el método `start(...)` tomo un `Stage` como parámetro. El gráfico siguiente muestra la estructura de cualquier aplicación JavaFX:



Fuente de la imagen: http://www.oracle.com

Es como una obra de teatro: El `Stage` (escenario) es el contenedor principal, normal-

mente una ventana con borde y los típicos botones para maximizar, minimizar o cerrar la ventana. Dentro del `Stage` se puede añadir una `Scene` (escena), la cual puede cambiarse dinámicamente por otra `Scene`. Dentro de un `Scene` se añaden los nodos JavaFX, tales como `AnchorPane`, `TextBox`, etc.
Para tener más información puedes consultar Working with the JavaFX Scene Graph.
Abre el archivo `MainApp.java` y reemplaza todo su código con el código siguiente:

```
1  package ch.makery.direcciones;
2
3  import java.io.IOException;
4
5  import javafx.application.Application;
6  import javafx.fxml.FXMLLoader;
7  import javafx.scene.Scene;
8  import javafx.scene.layout.AnchorPane;
9  import javafx.scene.layout.BorderPane;
10 import javafx.stage.Stage;
11
12 public class MainApp extends Application {
13         private Stage primaryStage;
14         private BorderPane rootLayout;
15
16         @Override
17         public void start(Stage primaryStage) {
18                 this.primaryStage = primaryStage;
19                 this.primaryStage.setTitle("Directorios App");
20
21                 initRootLayout();
22                 showPersonaOverview();
23         }
24         /**
25         * Inicializa el diseño raíz.
26         */
27         public void initRootLayout(){
28                 try{
29                         //Carga el diseño raíz del archivo fxml.
30                         FXMLLoader loader = new FXMLLoader();
31                         loader.setLocation(MainApp.class.getResource("view/RootLayout.fx
32                         rootLayout = (BorderPane) loader.load();
33                         //Muestra la escena que contiene el diseño raíz.
34                         Scene scene = new Scene(rootLayout);
35                         primaryStage.setScene(scene);
36                         primaryStage.show();
37                 }catch (IOException e) {
```

```
38                          e.printStackTrace();
39                  }
40          }
41          /**
42          * Muestra la descripción general de la persona dentro del diseño
    ↪  raíz.
43          */
44          public void showPersonaOverview(){
45                  try{
46                          //Carga datos de persona
47                          FXMLLoader loader = new FXMLLoader();
48                          loader.setLocation(MainApp.class.getResource("view/PersonaOvervi
49                          AnchorPane personaOverview = (AnchorPane)
                             ↪  loader.load();
50                          //Carga los datos de la persona en el centro del
                             ↪  diseño raíz.
51                          rootLayout.setCenter(personaOverview);
52                  }catch (IOException e) {
53                          e.printStackTrace();
54                  }
55          }
56          /**
57          * Retorna el escenario principal
58          * @return
59          */
60          public Stage getPriStage(){
61                  return primaryStage;
62          }
63          public static void main(String[] args) {
64                  launch(args);
65          }
66  }
```

Los diferentes comentarios deben darte pistas sobre lo que hace cada parte del código. Si ejecutas la aplicación ahora, verás algo parecido a la captura de pantalla incluida al principio de este artículo.

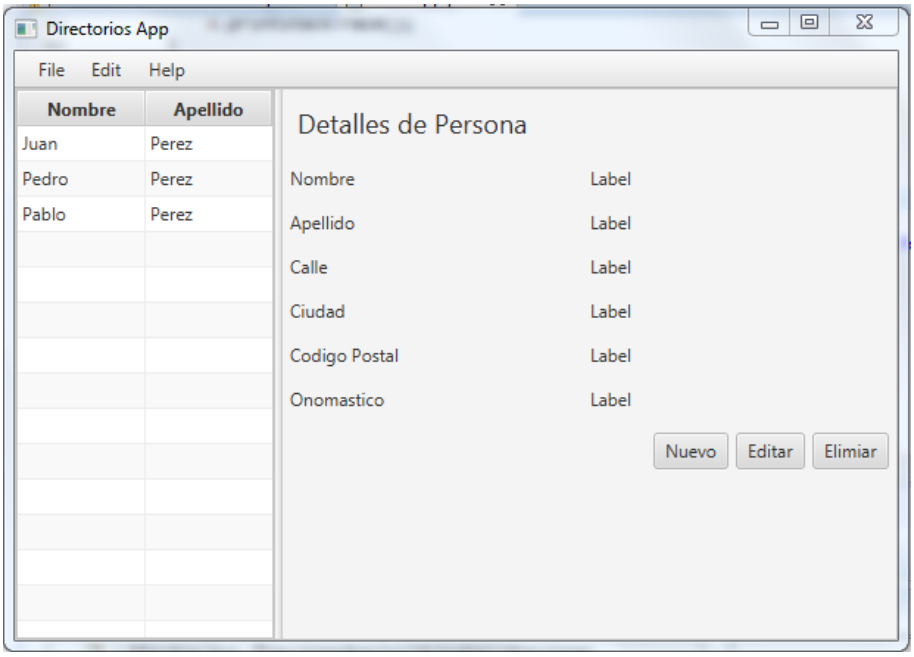
1.10. Problemas frecuentes

Si JavaFX no encuentra un archivo `fxml` puedes obtener el siguiente mensaje de error: `java.lang.IllegalStateException: Location is not set.`
Para resolverlo comprueba otra vez que no hayas escrito mal el nombre de tus archivos `fxml`.

Si todavía no te funciona, descárgate el código de esta parte del tutorial y pruébalo con el fxml proporcionado.

Parte 2

Modelo y TableView



2.1. Contenidos en Parte 2

- Creación de una clase para el modelo.
- Uso del modelo en una ObservableList.
- Visualización del modelo mediante TableView y Controladores.

2.2. Crea la clase para el Modelo

Neceistamos un modelo para contener la información sobre los contactos de nuestra agenda. Añade una nueva clase al paquete encargado de contener los modelos (`ch.makery.direcciones.model`

) denominada Persona. La clase Persona tendrá atributos (instancias de clase) para el nombre, la dirección y la fecha de nacimiento. Añade el código siguiente a la clase. Explicaré detalles de JavaFX después del código.
Persona.java

```java
package ch.makery.direcciones.model;

import java.time.LocalDate;

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

/**
* Modelo de la clase persona
* @author Mateo Mtz
*
*/
public class Persona {
        private StringProperty nombre;
        private StringProperty apellido;
        private StringProperty calle;
        private IntegerProperty codigoPostal;
        private StringProperty ciudad;
        private ObjectProperty<LocalDate> onomastico;
        public Persona(String nombre, String apellido){
                this.nombre = new SimpleStringProperty(nombre);
                this.apellido = new SimpleStringProperty(apellido);

                //
                this.calle = new SimpleStringProperty("Alguna calle");
                this.codigoPostal = new SimpleIntegerProperty(1234);
                this.ciudad = new SimpleStringProperty("Alguna ciudad");
                this.onomastico = new
                ↪  SimpleObjectProperty<LocalDate>(LocalDate.of(1999,  2,
                ↪  21));
        }
        public StringProperty getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
```

```
38                  this.nombre = new SimpleStringProperty(nombre);
39          }
40          public StringProperty getApellido() {
41                  return apellido;
42          }
43          public void setApellido(String apellido) {
44                  this.apellido = new SimpleStringProperty(apellido);
45          }
46          public StringProperty getCalle() {
47                  return calle;
48          }
49          public void setCalle(String calle) {
50                  this.calle = new SimpleStringProperty(calle);
51          }
52          public IntegerProperty getCodigoPostal() {
53                  return codigoPostal;
54          }
55          public void setCodigoPostal(int codigoPostal) {
56                  this.codigoPostal = new
                    ↪  SimpleIntegerProperty(codigoPostal);
57          }
58          public StringProperty getCiudad() {
59                  return ciudad;
60          }
61          public void setCiudad(String ciudad) {
62                  this.ciudad = new SimpleStringProperty(ciudad);
63          }
64          public ObjectProperty<LocalDate> getOnomastico() {
65                  return onomastico;
66          }
67          public void setOnomastico(LocalDate onomastico) {
68                  this.onomastico = new
                    ↪  SimpleObjectProperty<LocalDate>(onomastico);
69          }
70  }
```

2.2.1. Explicación del código

- Con JavaFX es habitual usar `Propiedades` para todos los atributos de un clase usada como modelo. Una `Propiedad` permite, entre otras cosas, recibir notificaciones automáticamente cuando el valor de una variable cambia (por ejemplo, si cambia `apellido`. Esto ayuda a mantener sincronizados la vista y los datos. Para aprender más sobre `Propiedades` lee Using JavaFX Properties and Binding.

- `LocalDate`, el tipo que usamos para especificar la fecha de nacimiento (`onomastico`) es parte de la nueva API de JDK 8 para la fecha y la hora.

2.3. Una lista de personas

Los principales datos que maneja nuestra aplicación son una colección de personas. Vamos a crear una lista de objetos de tipo `Persona` dentro de la clase principal `MainApp`. El resto de controladores obtendrá luego acceso a esa lista central dentro de `MainApp`.

2.3.1. Lista observable (ObservableList)

Estas clases gráficas de JavaFX que necesitan ser informadas sobre los cambios en la lista de personas. Esto es importante, pues de otro modo la vista no estará sincronizada con los datos. Para estos fines, JavaFX ha introducido nuevas clases de colección.
De esas colecciones, necesitamos la denominada `ObservableList`. Para crear una nueva `ObservableList`, añade el código siguiente al principio de la clase `MainApp`. También añadiremos un constructor para crear datos de ejemplo y un método de consulta (get) público:
`MainApp.java`

```
1         /**
2         * Lista de personas en una ObservableList
3         */
4         private ObservableList<Persona> personData =
          ↪  FXCollections.observableArrayList();
5         /**
6         * Constructor
7         */
8         public MainApp(){
9                 //Agregar algunas personas a la lista
10                personData.add(new Persona("Juan", "Perez"));
11                personData.add(new Persona("Pedro", "Perez"));
12                personData.add(new Persona("Pablo", "Perez"));
13        }
14        /**
15        * Devuelve los datos como una lista observable de personas.
16        * @return
17        */
18        public ObservableList<Persona> getPersonData(){
19                return personData;
20        }
```

2.4. PersonOverviewController

Finalmente vamos a añadir datos a nuestra table. Para ello necesitaremos un controlador específico para la vista `PersonOverview.fxml`.

1. Crea una clase normal dentro del paquete **view** denominado `PersonOverviewController.java`. (Debemos ponerlo en el mismo paquete que `PersonOverview.fxml` o el Scene Builder no lo encontrará, al menos no en la versión actual).

2. Añadiremos algunos atributos para acceder a la tabla y las etiquetas de la vista. Estos atributos irán precedidos por la anotación `@FXML`. Esto es necesario para que la vista tenga acceso a los atributos y métodos del controlador, incluso aunque sean privados. Una vez definida la vista en fxml, la aplicación se encargará de rellenar automáticamente estos atributos al cargar el fxml. Así pues, añade el código siguiente:

> **Nota:** acuérdate siempre de importar **javafx**, NO AWT ó Swing!.

```
1  package ch.makery.direcciones.view;
2
3  import ch.makery.direcciones.MainApp;
4  import ch.makery.direcciones.model.Persona;
5  import javafx.fxml.FXML;
6  import javafx.scene.control.Label;
7  import javafx.scene.control.TableColumn;
8  import javafx.scene.control.TableView;
9
10 public class PersonOverviewController {
11         @FXML
12         private TableView<Persona> personTable;
13         @FXML
14         private TableColumn<Persona, String> nombresColumna;
15         @FXML
16         private TableColumn<Persona, String> apellidosColumna;
17
18         @FXML
19         private Label nombreLabel;
20         @FXML
21         private Label apelidoLabel;
22         @FXML
23         private Label calleLabel;
24         @FXML
25         private Label codigoPostaLabel;
26         @FXML
27         private Label ciudadLabel;
```

```
28          @FXML
29          private Label onomasticoLabel;
30
31          //Referencia a la clase MainApp
32          private MainApp mainApp;
33          /**
34          * Constructor
35          * Se llama al constructor antes del método initialize()
36          */
37          public PersonOverviewController(){
38          }
39          /**
40          * Inicializa la clase de controlador
41          * Este método se llama automáticamente después de cargar el
    ↪  archivo fxml.
42          */
43          @FXML
44          private void initialize(){
45                  //Inicialice la tabla de personas con las dos columnas.
46                  nombresColumna.setCellValueFactory(cellData ->
                    ↪  cellData.getValue().getNombre());
47                  apellidosColumna.setCellValueFactory(cellData ->
                    ↪  cellData.getValue().getApellido());
48          }
49          /**
50          * Es llamado por la aplicación principal para devolverse una
    ↪  referencia a sí mismo.
51          * @param mainApp
52          */
53          public void setMainApp(MainApp mainApp){
54                  this.mainApp = mainApp;
55                  personTable.setItems(mainApp.getPersonData());
56          }
57  }
```

Este código necesita cierta explicación:

- Los campos y métodos donde el archivo fxml necesita acceso deben ser anotados con `@FXML`. En realidad, sólo si son privados, pero es mejor tenerlos privados y marcarlos con la anotación.
- El método `initialize()` es invocado automáticamente tras cargar el fxml. En ese momento, todos los atributos FXML deberían ya haber sido inicializados.
- El método `setCellValueFactory(...)` que aplicamos sobre las columnas de la tabla se usa para determinar qué atributos de la clase `Persona` deben ser usados para cada

columna particular. La flecha `->` indica que estamos usando una característica de Java 8 denominada *Lambdas*. Otra opción sería utilizar un `PropertyValueFactory`, pero entonces no ofrecería seguridad de tipo (*type-safe*).

2.4.1. Conexión de MainApp con PersonOverviewController

El método `setMainApp(...)` debe ser invocado desde la clase `MainApp`. Esto nos da la oportunidad de acceder al objeto `MainApp` para obtener la lista de `Persona` y otras cosas. Sustituye el método `showPersonOverview()` con el código siguiente, el cual contiene dos líneas adicionales:
`MainApp.java` - nuevo método `showPersonOverview()`

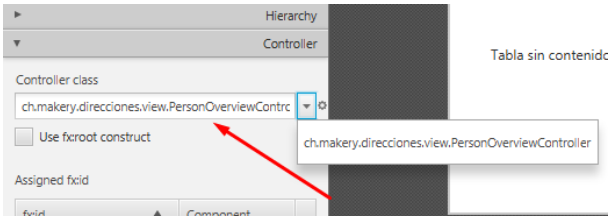
```
1         /**
2          * Muestra la descripción general de la persona dentro del diseño
   ↪  raíz.
3          */
4         public void showPersonaOverview(){
5                 try{
6                         //Carga datos de persona
7                         FXMLLoader loader = new FXMLLoader();
8                         loader.setLocation(MainApp.class.getResource("view/PersonaOvervi
9                         AnchorPane personaOverview = (AnchorPane)
                          ↪  loader.load();
10
11                        //Carga los datos de la persona en el centro del
                          ↪  diseño raíz.
12                        rootLayout.setCenter(personaOverview);
13
14                        //Darle al controlador acceso a la App - lineas
                          ↪  adicionales
15                        PersonOverviewController controller =
                          ↪  loader.getController();
16                        controller.setMainApp(this);
17                }catch (IOException e) {
18                        e.printStackTrace();
19                }
20        }
```

2.5. Vincular la vista al controlador

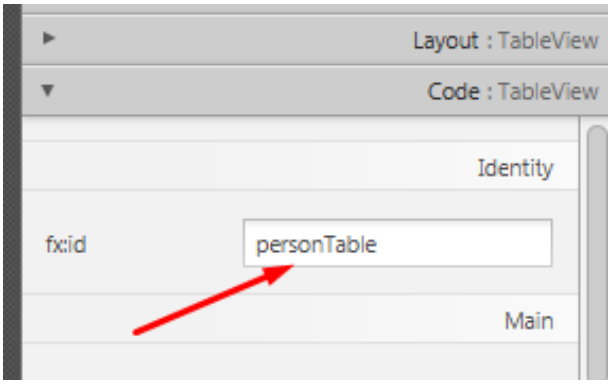
¡Ya casi lo tenemos! Pero falta un detalle: no le hemos indicado a la vista declarada en `PersonOverview.fxml` cuál es su controlador y que elemento hacer corresponder to cada uno de los atributos en el controlador.

1. `PersonOverview.fxml` en *SceneBuilder*.

2. Abre la sección *Controller* en el lado izquierdo y selecciona `PersonOverviewController` como **controlador**.

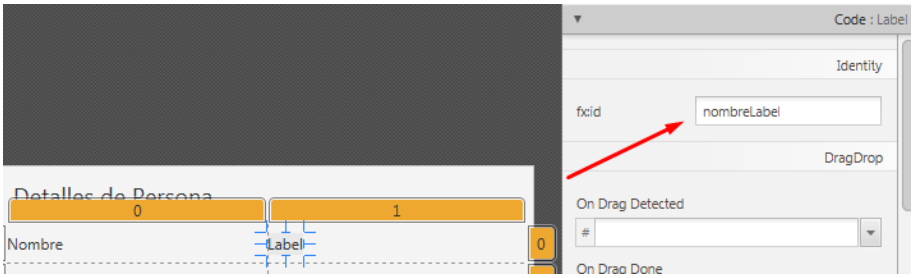


3. Selecciona `TableView` en la sección *Hierarchy* y en el apartado *Code* escribe o selecciona `personTable` en la propiedad **fx:id**.



4. Haz lo mismo para las columnas, poniendo `nombre` y `apellido` con sus **fx:id** respectivamente.

5. Para **cada etiqueta** en la segunda columna, introduce el **fx:id** que corresponda.



6. Importante: En Eclipse **refresca el projecto DireccionesApp** (tecla F5). Esto es necesario porque a veces Eclipse no se da cuenta de los cambios realizados desde el *Scene Builder*.

2.6. Inicia la aplicación

Ahora, cuando ejecutes la aplicación, deberías obtener algo parecido a la captura de pantalla incluida al principio de este artículo.

Enhorabuena!

Parte 3

Interacción con el usuario

3.1. Contenidos en parte 3

- Respuesta a cambios en la selección dentro de la tabla.
- Añade funcionalidad de los botones añadir, editar, y borrar.
- Crear un diálogo emergente (popup dialog) a medida para editar un contacto.
- Validación de la entrada del usuario.

3.2. Respuesta a cambios en la selección de la Tabla

Todavía no hemos usado la parte derecha de la interfaz de nuestra aplicación. La intención es usar esta parte para mostrar los detalles de la persona seleccionada por el usuario en la tabla.
En primer lugar vamos a añadir un nuevo método dentro de `PersonOverviewController` que nos ayude a rellenar las etiquetas con los datos de una sola persona.

Crea un método llamado `showPersonaDetails(Persona persona)` . Este método recorrerá todas las etiquetas y establecerá el texto con detalles de la persona usando `setText(...)` . Si en vez de una instancia de `Person` se pasa null entonces las etiquetas deben ser borradas.

`PersonOverviewController.java` nuevo método `showPersonaDetails(Persona persona)`

```
1     /**
2     * Rellena todos los campos de texto para mostrar detalles sobre la
  ↪   persona.
3     * Si la persona especificada es nula, se borran todos los campos de
  ↪   texto.
4     * @param persona
5     */
6    private void showPersonaDetails(Persona persona){
```

```
7         if(persona != null){
8             //Rellene las etiquetas con información del objeto persona.
9             nombreLabel.setText(persona.getNombre().get());
10            apelidoLabel.setText(persona.getApellido().get());
11            calleLabel.setText(persona.getCalle().get());
12            codigoPostaLabel.setText(persona.getCodigoPostal().get() + "");
13            ciudadLabel.setText(persona.getCiudad().get());
14            //¡Necesitamos una forma de convertir el cumpleaños en una
              ↪  Cadena!
15            onomasticoLabel.setText(...);
16            //
              ↪  onomasticoLabel.setText(DateUtil.format(persona.getOnomastico().get()))
17        }else {
18            //Si la persona es nula, quitar todo el texto
19            nombreLabel.setText("");
20            apelidoLabel.setText("");
21            calleLabel.setText("");
22            codigoPostaLabel.setText("");
23            ciudadLabel.setText("");
24            onomasticoLabel.setText("");
25        }
26    }
```

3.3. Convierte la fecha de nacimiento en una cadena

Te darás cuenta de que no podemos usar el atributo `onomastico` directamente para establecer el valor de una `Label` porque se requiere un `String`, y `onomastico` es de tipo `LocalDate`. Así pues necesitamos convertir `onomastico` de `LocalDate` a `String`.

En la práctica vamos a necesitar convertir entre `LocalDate` y `String` en varios sitios y en ambos sentidos. Una buena práctica es crear una clase auxiliar con métodos estáticos (`static`) para esta finalidad. Llamaremos a esta clase `DateUtil` y la ubicaremos una paquete separado denominado `ch.makery.direcciones.util` :

`DateUtil.java`

```
1     package ch.makery.direcciones.util;
2
3     import java.time.LocalDate;
4     import java.time.format.DateTimeFormatter;
5
6     /**
7      * Funciones de ayuda para manejar fechas.
```

```
8       * @author Marco Jakob
9       *
10      */
11     public class DateUtil {
12         /**
13          * El patrón de fecha que se utiliza para la conversión.
14          * Cambialo como quieras.
15          */
16         private static final String DATE_PATTERN = "dd-MM-yyyy";
17         /**
18          * Formateado de fecha
19          */
20         private static final DateTimeFormatter DATE_FORMATTER =
21                 DateTimeFormatter.ofPattern(DATE_PATTERN);
22         /**
23          * Devuelve la fecha que se pasó como parametro en una cadena
24          * @param date fecha en formato LocalDate
25          * @return fecha en un String
26          */
27         public static String format(LocalDate date){
28             if(date == null){
29                 return null;
30             }
31             return DATE_FORMATTER.format(date);
32         }
33         /**
34          * Convierte una cadena en formato de fecha a un objeto LocalDate
35          * Retorna nulo si la cadena no pudo convertirse
36          *
37          * @param dateString fecha como un String
38          * @return un objeto de tipo fecha, si no se pudo convertir
    ↪  retorna nulo
39          */
40         public static LocalDate parse(String dateString){
41             try {
42                 return DATE_FORMATTER.parse(dateString, LocalDate::from);
43             } catch (Exception e) {
44                 return null;
45             }
46         }
47         /**
48          * Comprueba la cadena si es una fecha valida
49          * @param dateString
50          * @return verdadero si la cadena es una fecha valida.
51          */
```

```
52          public static boolean validDate(String dateString){
53              //Intenta analizar la cadena
54              return DateUtil.parse(dateString) != null;
55          }
56      }
```

3.3.1. Utilización de la clase DateUtil

Ahora necesitamos utilizar la nueva clase `DateUtil` en el método `showPersonDetails` de `PersonaOverviewController`. Sustituye el TODO que habíamos añadido con la línea siguiente:

```
1       onomasticoLabel.setText(DateUtil.format(persona.getOnomastico().get()));
```

3.4. Detecta cambios de selección en la tabla

Para enterarse de que el usuario ha seleccionado a un persona en la tabla de contactos, necesitamos escuchar los cambios. Esto se consigue mediante la implementación de un interface de JavaFX que se llama `ChangeListener` con un método llamado `changed(...)`. Este método solo tiene tres parámetros: `observable`, `oldValue`, y `newValue`.
En Java 8 la forma más elegante de implementar una interfaz con un único método es mediante una *lambda expression*. Añadiremos algunas líneas al método `initialize()` de `PersonaOverviewController`. El código resultante se asemejará al siguiente:
`PersonOverviewController.java`

```
1       /**
2            * Inicializa la clase de controlador
3            * Este método se llama automáticamente después de cargar el
    ↪   archivo fxml.
4            */
5           @FXML
6           private void initialize(){
7                   //Inicialice la tabla de personas con las dos columnas.
8                   nombresColumna.setCellValueFactory(cellData ->
                    ↪   cellData.getValue().getNombre());
9                   apellidosColumna.setCellValueFactory(cellData ->
                    ↪   cellData.getValue().getApellido());
10
11                  //Limpiar detalles de persona
12                  showPersonaDetails(null);
13
```

```
14                  //Escuche los cambios de selección y muestre los detalles
                    ↪  de la persona cuando cambie.
15                  personTable.getSelectionModel().selectedItemProperty().addListener(
16                                  (observable, oldValue, newValue) ->
                                    ↪  showPersonaDetails(newValue));
17          }
```

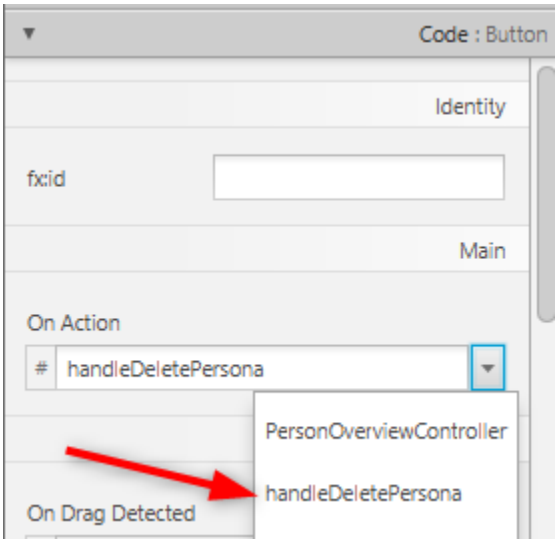
Con `showPersonDetails(null);` borramos los detalles de una persona.
Con `personaTable.getSelectionModel...` obtenemos la *selectedItemProperty* de la tabla de personas, y le añadimos un *listener*. Cuando quiera que el usuario seleccione a una persona en la tabla, nuestra *lambda expression* será ejecutada: se toma la persona recien seleccionada y se le pasa al método `showPersonDetails(...)` .
Intenta **ejecutar tu aplicación** en este momento. Comprueba que cuando seleccionas a una persona, los detalles sobre esta son mostrados en la parte derecha de la ventana.
Si algo no funciona, puedes comparar tu clase `PersonaOverviewController` con PersonOverviewController.java.

3.5. El botón de borrar (Delete)

Nuestro interfaz de usuario ya contiene un botón de borrar, pero sin funcionalidad. Podemos seleccionar la acción a ejecutar al pulsar un botón desde el *Scene Builder*. Cualquier método de nuestro controlador anotado con `@FXML` (o declarado como *public*) es accesible desde *Scene Builder*. Así pues, empecemos añadiendo el método de borrado al final de nuestra clase `PersonaOverviewController` :
`PersonaOverviewController.java`

```
1       /**
2            * Se llama cuando el usuario hace clic en el botón Eliminar.
3            */
4           @FXML
5           private void handleDeletePersona(){
6                   int selectedIndex =
                    ↪  personTable.getSelectionModel().getSelectedIndex();
7                   personTable.getItems().remove(selectedIndex);
8           }
```

Ahora, abre el archivo `PersonOverview.fxml` en el *SceneBuilder*. Selecciona el botón *Eliminar*, abre el apartado *Code* y pon `handleDeletePerson` en el menú desplegable denominado **On Action**.



3.6. Gestión de errores

Si ejecutas tu aplicación en este punto deberías ser capaz de borrar personas de la tabla. Pero, ¿qué ocurre si pulsas el botón de borrar sin seleccionar a nadie en la tabla.
Se produce un error de tipo `ArrayIndexOutOfBoundsException` porque no puede borrar una persona en el índice `-1` , que es el valor devuelto por el método `getSelectedIndex()` - cuando no hay ningún elemento seleccionado.
Ignorar semejante error no es nada recomendable. Deberíamos hacerle saber al usuario que tiene que seleccionar una persona previamente para poderla borrar (incluso mejor sería deshabilitar el botón para que el usuario ni siquiera tenga la oportunidad de realizar una acción incorrecta).

Con algunos cambios en el método `handleDeletePerson()` podemos mostrar una simple ventana de diálogo emergente en el caso de que el usuario pulse el botón Delete sin haber seleccionado a nadie en la tabla de contactos:
`PersonOverviewController`

```
1     /**
2          * Se llama cuando el usuario hace clic en el botón Eliminar.
3          */
4         @FXML
5         private void handleDeletePersona(){
6                 int selectedIndex =
                  ↪  personTable.getSelectionModel().getSelectedIndex();
7                 if(selectedIndex >= 0){
8                         personTable.getItems().remove(selectedIndex);
9                 }
10                else {
11                        Alert alert = new Alert(AlertType.WARNING);
```

```
12                          alert.setTitle("Sin seleccion");
13                          alert.setHeaderText("No has seleccionado una
                            ↪  persona");
14                          alert.setContentText("Por favor selecciona una
                            ↪  persona de la lista para eliminar");
15                          alert.showAndWait();
16                  }
17          }
```

Nota: Para ver más ejemplos de utilización de ventanas de diálogo, consulta JavaFX 8 Dialogs.

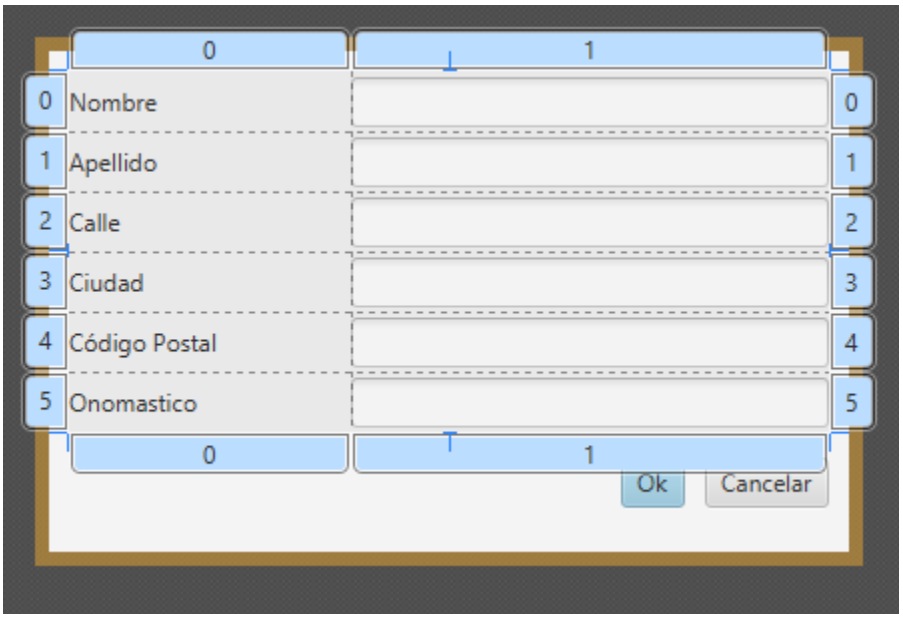
3.7. Diálogos para crear y editar contactos

Las acciones de editar y crear nuevo contacto necesitan algo más de elaboración: vamos a necesitar una ventana de diálogo a medida (es decir, un nuevo `stage`) con un formulario para preguntar al usuario los detalles sobre la persona.

3.7.1. Diseña la ventana de diálogo

1. Crea un nuevo archivo fxml llamado `PersonEditDialog.fxml` dentro del paquete *view*.

2. Usa un panel de rejilla (`GridPane`), etiquetas (`Label`), campos de texto (`TextField`) y botones (Button) para crear una ventana de diálogo como la siguiente:



3.7.2. Crear el controlador

Crea el controlador para la ventana de edición de personas y llámalo `PersonEditDialogController.ja`
:
`PersonEditDialogController`

```
    package ch.makery.direcciones.view;

import ch.makery.direcciones.model.Persona;
import ch.makery.direcciones.util.DateUtil;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.TextField;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.image.Image;
import javafx.stage.Stage;

/**
 * Diálogo para editar detalles de una persona
 * @author Yo
 *
 */
public class PersonEditDialogController {
        @FXML
        private TextField nombreField;
        @FXML
        private TextField apellidoField;
        @FXML
        private TextField calleField;
        @FXML
        private TextField codigoPostalField;
        @FXML
        private TextField ciudadField;
        @FXML
        private TextField onomasticoField;

        private Stage dialogStage;
        private Persona persona;
        private boolean okClicked = false;
        /**
         * Inicializa la clase de controlador.
         * Este método se llama automáticamente después de cargar el
 ↪  archivo fxml.
         */
        @FXML
```

```
39          private void initialize(){
40
41          }
42          /**
43           * Establece el escenario de este diálogo.
44           * @param dialogStage
45           */
46          public void setDialogStage(Stage dialogStage){
47                  this.dialogStage = dialogStage;
48                  this.dialogStage.getIcons().add(new
                     ↪  Image("file:resources/images/contacto.png"));
49          }
50          /**
51           * Establece la persona a editar en el diálogo.
52           * @param persona
53           */
54          public void setPerson(Persona persona){
55                  this.persona = persona;
56                  nombreField.setText(persona.getNombre().get());
57                  apellidoField.setText(persona.getApellido().get());
58                  calleField.setText(persona.getCalle().get());
59                  codigoPostalField.setText(persona.getCodigoPostal().get() +
                     ↪  "");
60                  ciudadField.setText(persona.getCiudad().get());
61                  onomasticoField.setText(DateUtil.format(persona.getOnomastico().get()));
62                  onomasticoField.setPromptText("dd-mm-yyyy");
63          }
64          /**
65           * Devuelve verdadero si el usuario hizo clic en aceptar
66           * falso de lo contrario
67           * @return
68           */
69          public boolean isOkClicked(){
70                  return okClicked;
71          }
72          /**
73           * Valida la entrada del usuario en los campos de texto.
74           * @return verdadero si la entrada es valida.
75           */
76          private boolean isInputValid(){
77                  String errorMessage = "";
78
79          if (nombreField.getText() == null || nombreField.getText().length()
            ↪  == 0) {
80              errorMessage += "El nombre no es valido!\n";
```

```
81          }
82          if (apellidoField.getText() == null ||
            ↪  apellidoField.getText().length() == 0) {
83              errorMessage += "El apellido no es valido!\n";
84          }
85          if (calleField.getText() == null || calleField.getText().length()
            ↪  == 0) {
86              errorMessage += "La calle no es valido!\n";
87          }
88
89          if (codigoPostalField.getText() == null ||
            ↪  codigoPostalField.getText().length() == 0) {
90              errorMessage += "El código postal no es valido!\n";
91          } else {
92              // Intenta analizar el código postal en un int
93              try {
94                  Integer.parseInt(codigoPostalField.getText());
95              } catch (NumberFormatException e) {
96                  errorMessage += "El Código Postal no es valido, debe se un
                    ↪  entero!\n";
97              }
98          }
99
100         if (ciudadField.getText() == null || ciudadField.getText().length()
            ↪  == 0) {
101             errorMessage += "No valid city!\n";
102         }
103
104         if (onomasticoField.getText() == null ||
            ↪  onomasticoField.getText().length() == 0) {
105             errorMessage += "Onomastico no es valido!\n";
106         } else {
107             if (!DateUtil.validDate(onomasticoField.getText())) {
108                 errorMessage += "El onomastico no es valido. Usa el formato
                    ↪  dd-mm-yyyy!\n";
109             }
110         }
111
112         if (errorMessage.length() == 0) {
113             return true;
114         } else {
115             // Mostrar mensaje de error.
116                 Alert alert = new Alert(AlertType.WARNING);
117                         alert.setTitle("Campos no validos");
```

```
118                         alert.setHeaderText("Por favor corrige los campos no
                            ↪  validos");
119                         alert.setContentText(errorMessage);
120                         alert.showAndWait();
121             return false;
122         }
123         }
124         /**
125          * Se llama cuando el usuario hace clic en Aceptar
126          */
127         @FXML
128         private void handleOk(){
129                 if(isInputValid()){
130                         persona.setNombre(nombreField.getText());
131                         persona.setApellido(apellidoField.getText());
132                         persona.setCalle(calleField.getText());
133                         persona.setCodigoPostal(Integer.parseInt(codigoPostalField.getTe
134                         persona.setCiudad(ciudadField.getText());
135                         persona.setOnomastico(DateUtil.parse(onomasticoField.getText()))
136                         okClicked = true;
137                         dialogStage.close();
138                 }
139         }
140         /**
141          * Se llama cuando el usuario hace clic en cancelar.
142          */
143         @FXML
144         private void handleCancel(){
145                 dialogStage.close();
146         }
147 }
148
149         }
```

Algunas cuestiones relativas a este controlador:

- El método `setPerson(...)` puede ser invocado desde otra clase para establecer la persona que será editada.
- Cuando el usuario pula el botón OK, el método `handleOk()` es invocado. Primero se valida la entrada del usuario mediante la ejecución del método `isInputValid()`. Sólo si la validación tiene éxito el objeto persona es modificado con los datos introducidos por el usuario. Esos cambios son aplicados directamente sobre el objeto pasado como argumento del método `setPerson(...)` !

- El método boleano `okClicked` se utiliza para determinar si el usuario ha pulsado el botón OK o el botón Cancelar.

3.7.3. Enlaza la vista y el controlador

Una vez creadas la vista (FXML) y el controlador, necesitamos vincular el uno con el otro:

1. Abre el archivo `PersonEditDialog.fxml` .
2. En la sección Controller a la izquierda selecciona `PersonEditDialogController` como clase de control.
3. Establece el campo **fx:id** de todas los `TextField` con los identificadores de los atributos del controlador correspondientes.
4. Especifica el campo **onAction** de los dos botones con los métodos del controlador correspondientes a cada acción.

3.7.4. Abriendo la ventana de diálogo

Añade un método para cargar y mostrar el método de edición de una persona dentro de la clase `MainApp` : `MainApp.java`

```
1     /**
2          * Abre un cuadro de diálogo para editar detalles para la persona
  ↪   especificada.
3          * Si el usuario hace clic en Aceptar, los cambios se guardan en
  ↪   el objeto
4          * de persona proporcionado y se devuelve verdadero.
5          *
6          * @param persona objeto persona a editar
7          * @return  verdadero si el usuario hizo clic en Aceptar, falso en
  ↪   caso contrario.
8          */
9         public boolean showPersonEditDialog(Persona persona){
10                try {
11                        //Cargue el archivo fxml y cree una nueva escena
                           ↪  para el
12                        //cuadro de diálogo emergente.
13                        FXMLLoader loader = new FXMLLoader();
14                        loader.setLocation(
15                                        MainApp.class.getResource("view/PersonEditDialog
16                                        );
17                        AnchorPane page = (AnchorPane) loader.load();
18
```

```
                        //Crear el cuadro de diálogo de la escena
                        Stage dialogStage = new Stage();
                        dialogStage.setTitle("Editar persona");
                        dialogStage.initModality(Modality.WINDOW_MODAL);
                        dialogStage.initOwner(primaryStage);
                        Scene scene = new Scene(page);
                        dialogStage.setScene(scene);

                        //Envia la persona al controlador
                        PersonEditDialogController controller =
                        ↪  loader.getController();
                        controller.setDialogStage(dialogStage);
                        controller.setPerson(persona);

                        //Muestra el diálogo y espera hasta que el usuario
                        ↪  lo cierre.
                        dialogStage.showAndWait();

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

Añade los siguientes métodos a la clase `PersonOverviewController` . Esos métodos llamarán al método `showPersonEditDialog(...)` desde `MainApp` cuando el usuario pulse en los botones Nuevo o Editar.
`PersonOverviewController.java`

```
    /**
         * Se llama cuando el usuario hace clic en el botón Nuevo.
         * Abre un cuadro de diálogo para editar detalles para una nueva
↪  persona.
         */
        @FXML
        private void handleNewPerson(){
                Persona temPersona = new Persona();
                boolean okClicked =
                ↪  mainApp.showPersonEditDialog(temPersona);
                if(okClicked){
                        mainApp.getPersonData().add(temPersona);
                }
        }
```

```
13          /*
14           * Se llama cuando el usuario hace clic en el botón editar.
15           * Abre un cuadro de diálogo para editar detalles para la persona
    ↪  seleccionada.
16           */
17          @FXML
18          private void handleEditPerson(){
19                  Persona personaSeleccionada =
                    ↪  personTable.getSelectionModel().getSelectedItem();
20                  if(personaSeleccionada != null){
21                          boolean okClicked =
                            ↪  mainApp.showPersonEditDialog(personaSeleccionada);
22                          if(okClicked){
23                                  showPersonaDetails(personaSeleccionada);
24                          }
25                  }else {
26                          Alert alert = new Alert(AlertType.WARNING);
27                          alert.setTitle("Sin seleccion");
28                          alert.setHeaderText("No has seleccionado una
                            ↪  persona");
29                          alert.setContentText("Por favor selecciona una
                            ↪  persona de la lista para eliminar");
30                          alert.showAndWait();
31                  }
32          }
```

Abre el archivo `PersonOverview.fxml` mediante *Scene Builder*. Elige los métodos correspondientes en el campo **On Action** para los botones Nuevo y Editar.

3.8. ¡Ya está!

Llegados a este punto deberías tener una aplicación de *libreta de contactos* en funcionamiento. Esta aplicación es capaz de añadir, editar y borrar personas. Tiene incluso algunas capacidades de validación para evitar que el usuario introduzca información incorrecta.
Espero que los conceptos y estructura de esta aplicación te permitan empezar tu propia aplicación JavaFX. ¡Disfruta !

Parte 4

Hojas de estilo CSS

4.1. Contenidos en Parte 4

- Estilos mediante CSS
- Añadiendo un Icono de Aplicación

4.2. Estilos mediante CSS

En JavaFX puedes dar estilo al interfaz de usuario utilizando hojas de estilo en cascada (CSS). ¡Esto es estupendo! Nunca había sido tan fácil personalizar la apariencia de una aplicación Java.
En este tutorial vamos a crear un tema oscuro (DarkTheme) inspirado en el diseño de Windows 8 Metro. El código CSS de los botones está basado en el artículo de blog JMetro - Windows 8 Metro controls on Java por Pedro Duque Vieira.

4.2.1. Familiarizándose con CSS

Para poder aplicar estilos a una aplicación JavaFX application debes tener una comprensión básica de CSS en general. Un buen punto de partida es este tutorial: CSS tutorial.
Para información más específica de CSS y JavaFX puedes consultar:

- Skinning JavaFX Applications with CSS - Tutorial de Oracle
- JavaFX CSS Reference - Referencia oficial

4.2.2. Estilo por defecto en JavaFX

Los estilos por defecto de JavaFX 8 se encuentran en un archivo denominado `modena.css` . Este archivo CSS se encuentra dentro del archivo jar `jfxrt.jar` que se encuentra en tu directorio de instalación de Java, en la ruta `/jdk1.8.x/jre/lib/ext/jfxrt.jar` .
Puedes descomprimir `jfxrt.jar` o abrirlo como si fuera un zip. Encontrarás el archivo `modena.css` en la ruta `com/sun/javafx/scene/control/skin/modena/`

Este estilo se aplica siempre a una aplicación JavaFX. Añadiendo un estilo personal podemos reescribir los estilos por defecto definidos en `modena.css` .

> **Truco:** Ayuda consultar el archivo CSS por defecto para ver qué estilos necesitas sobreescribir.

4.2.3. Vinculando hojas de estilo CSS

Añade un archivo CSS denominado `DarkTheme.css` al paquete *view*.
`DarkTheme.css`

A continuación, necesitamos vincular el CSS a nuestra escena. Podemos hacer esto programáticamente, mediante código Java, pero en esta ocasión vamos a utilizar *Scene Builder* para añadirlo a nuestros archivos FXML:

Añade el CSS a RootLayout.fxml

1. Abre el archivo `RootLayout.fxml` en *Scene Builder*.
2. Selecciona el `BorderPane` raíz en la sección *Hiperarchy*. En la vista *Properties* añade el archivo `DarkTheme.css` como hoja de estilo (campo denominado `Stylesheets`).

Añade el CSS a PersonEditDialog.fxml

1. Abre el archivo `PersonEditDialog.fxml` en *Scene Builder*. Selecciona el `AnchorPane` raíz e incluye `DarkTheme.css` como hoja de estilo en la sección *Properties*.
2. El fondo todavía es blanco, hay que añadir la clase `background` al `AnchorPane` raíz.
3. Selecciona el botón OK y elige *Default Button* en la vista *Properties*. Eso cambiará su color y lo convertirá en el botón "por defecto", el que se ejecutará si el usuario aprieta la tecla *enter*.

Añade el CSS a PersonOverview.fxml

1. Abre el archivo `PersonOverview.fxml` en *Scene Builder*. Selecciona el `AnchorPane` raíz en la sección *Hierarchy* y añade `DarkTheme.css` a sus `Stylesheets` .
2. A estas alturas deberías haber observado algunos cambios: La tabla y los botones son negros. Las clases de estilo `.table-view` y `.button` de `modena.css` se aplican automáticamente a la tabla y los botones. Ya que hemos redefinido (sobreescrito) algunos de esos estilos en nuestro propio CSS, los nuevos estilos se aplican automáticamente.
3. Posiblemente tengas que ajustar el tamaño de los botones para que se muestre todo el texto.
4. Selecciona el panel `AnchorPane` de la derecha, dentro del `SplitPane` .

5. Ves a la vista *Properties* y elige `background` como clase de estilo. El fondo debería volverse negro.

Etiquetas con un estilo diferente

En este momento, todas las etiquetas en el lado derecho tienen el mismo tamaño. Ya tenemos definidos en el CSS unos estilos denominados `.label-header` y `.label-bright` que vamos a usar para personalizar la apariencia de las etiquetas.

1. Selecciona la etiqueta (`Label`) *Detalle de personas* y añade `label-header` como clase de estilo.

2. A cada etiqueta en la columna de la derecha (donde se muestran los detalles de una persona), añade la clase de estilo `label-bright` .

4.3. Añadiendo un icono a la aplicación

Ahora mismo nuestra aplicación utiliza el icono por defecto para la barra de título y la barra de tareas:
Quedaría mucho mejor con un icono propio:

4.3.1. El archivo de icono

Un posible sitio para obtener iconos gratuitos es Icon Finder. Yo por ejemplo descargué este icono de libreta de direcciones.
Crea una carpeta dentro de tu aplicación llamado **resources** y añádele una subcarpeta para almacenar imágenes, llámala **images**. Pon el icono que hayas elegido dentro de la carpeta de imágenes. La estructura de directorios de tu carpeta debe tener un aspecto similar a este:

4.3.2. Establece el icono de la escena principal

Para establecer el icono de nuestra escena debemos añadir la línea de código siguiente al método `start(...)` dentro de `MainApp.java` .
`MainApp.java`
El método `start(...)` debería verse así ahora:
También puedes añadir un icono a la escena que contiene la venta de edición de los detalles de una persona.