

Tutorial JavaFX

Libreta de direcciones con JavaFX



[Enlace del proyecto original](#)

Mayo 2020

Índice general

1. Eclipse y Scene Builder	4
1.1. Contenidos en Parte 1	4
1.2. Prerrequisitos	4
1.3. Configuración de Eclipse	4
1.4. Enlaces útiles	6
1.5. Crea un nuevo proyecto JavaFX	6
1.5.1. Crea los paquetes	7
1.6. Crea el archivo FXML de diseño	8
1.7. Diseño mediante Scene Builder	9
1.8. Crea la aplicación principal	15
1.9. La clase principal en JavaFX	17
1.10. Problemas frecuentes	20
2. Modelo y TableView	22
2.1. Contenidos en Parte 2	22
2.2. Crea la clase para el Modelo	22
2.2.1. Explicación del código	24
2.3. Una lista de personas	25
2.3.1. Lista observable (ObservableList)	25
2.4. PersonOverviewController	26
2.4.1. Conexión de MainApp con PersonOverviewController	28
2.5. Vincular la vista al controlador	28
2.6. Inicia la aplicación	30
3. Interacción con el usuario	31
3.1. Contenidos en parte 3	31
3.2. Respuesta a cambios en la selección de la Tabla	31
3.3. Convierte la fecha de nacimiento en una cadena	32

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 actualizaciones 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 novatos, 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 proyecto 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

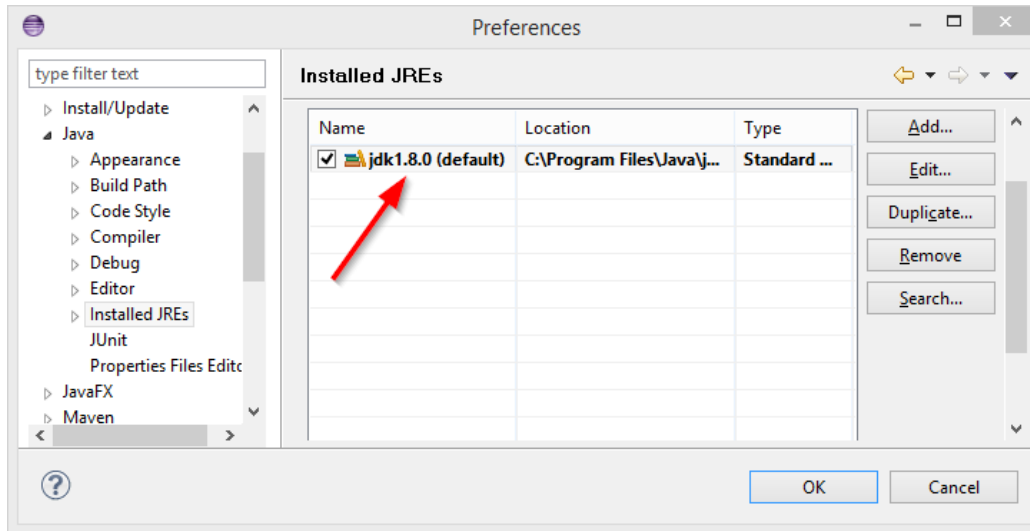
- Última versión de [Java JDK 8 o superior](#).
- Eclipse 4.3 o superior con el plugin e(fx)clipse. La forma más sencilla de obtenerlo es descargarse la distribución preconfigurada desde [e\(fx\)clipse website](#). Como alternativa puedes usar un [sitio de actualización](#) para tu instalación de Eclipse.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

1.3. Configuración de Eclipse

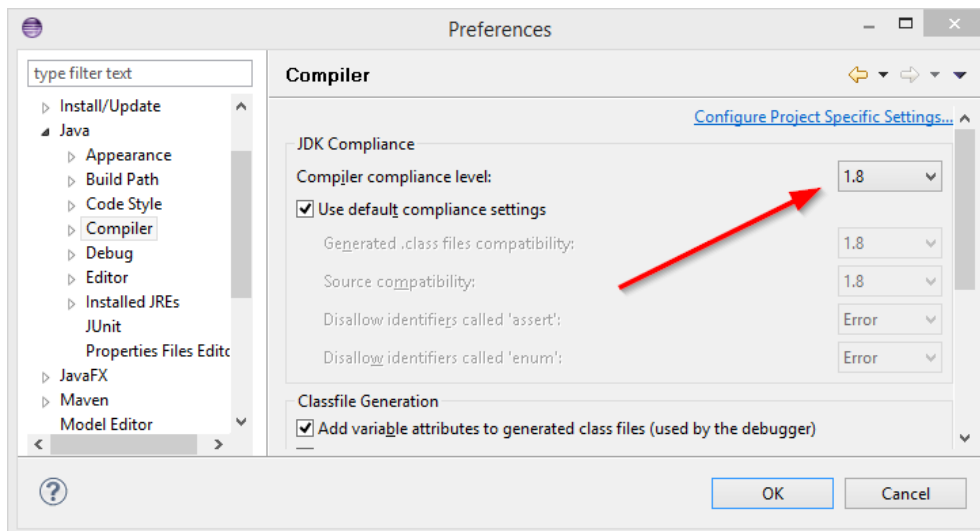
Hay que indicarle a Eclipse que use JDK 8 y también dónde se encuentra el ejecutable del Scene Builder:

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.

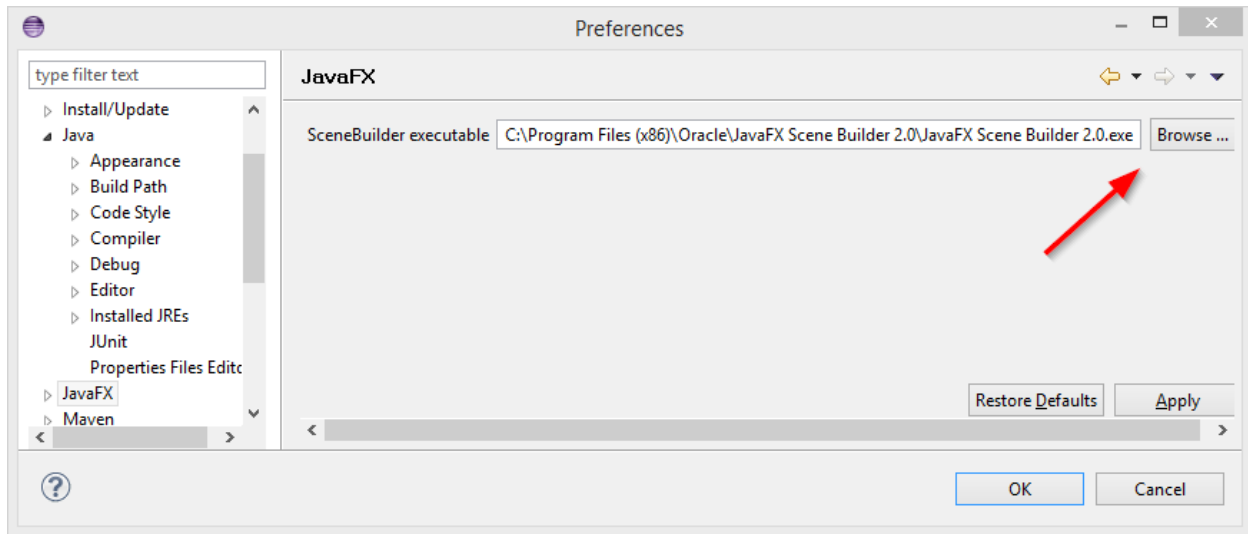
3. Elimina otros JREs o JDKs de tal manera que **JDK 8** se convierta en la opción por defecto.



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

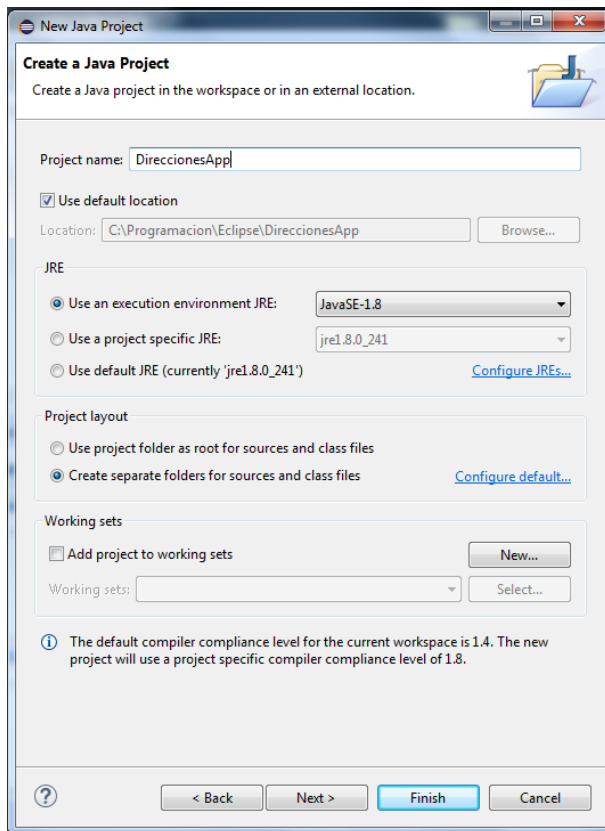
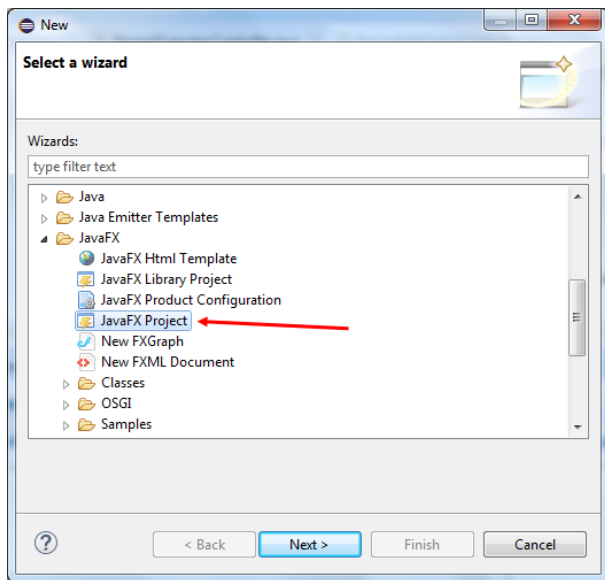
Te podría interesar los siguientes enlaces:

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*:

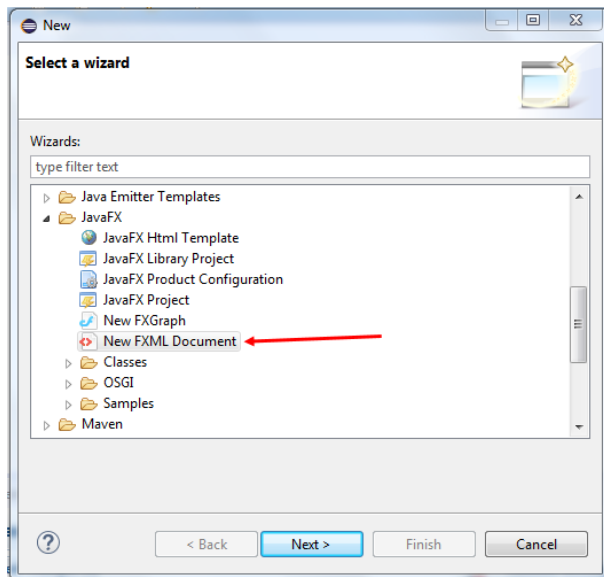
- `ch.makery.direcciones` - contendrá la mayoría de clases de control (C).
- `ch.makery.direcciones.model` - contendrá las clases del modelo (M).
- `ch.makery.direcciones.view` - contendrá las vistas (V)

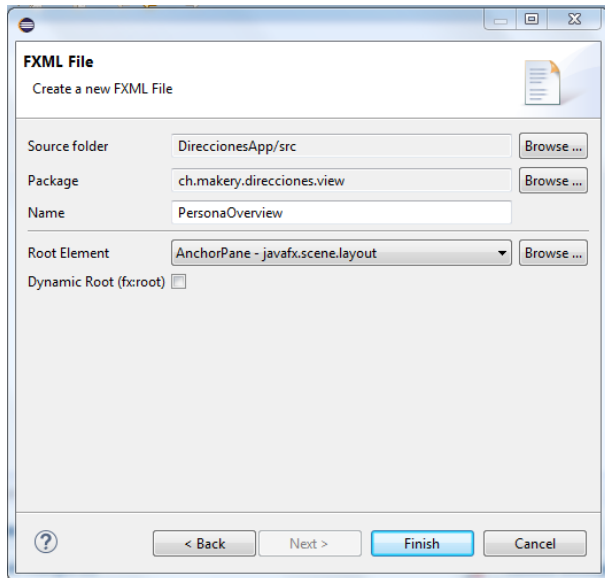
Nota: Nuestro paquete dedicado a las vistas contendrá también algunos controladores dedicados exclusivamente a una vista. Les llamaremos **controladores-vista**.

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`.

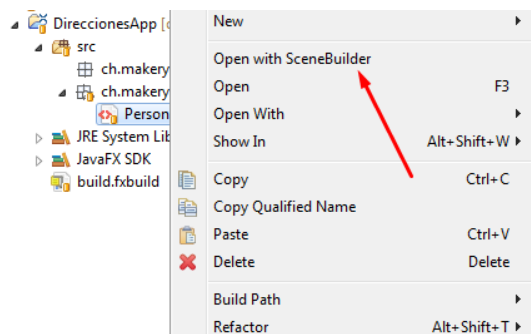




1.7. Diseño mediante Scene Builder

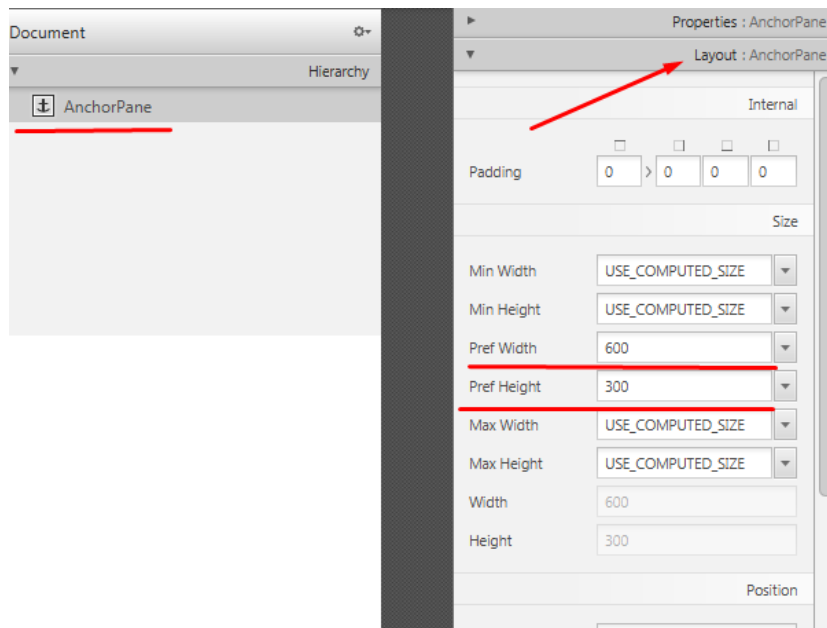
Nota: Si no puedes hacerlo funcionar, descarga las fuentes para esta parte del tutorial e inténtalo con el archivo fxml incluido.

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).

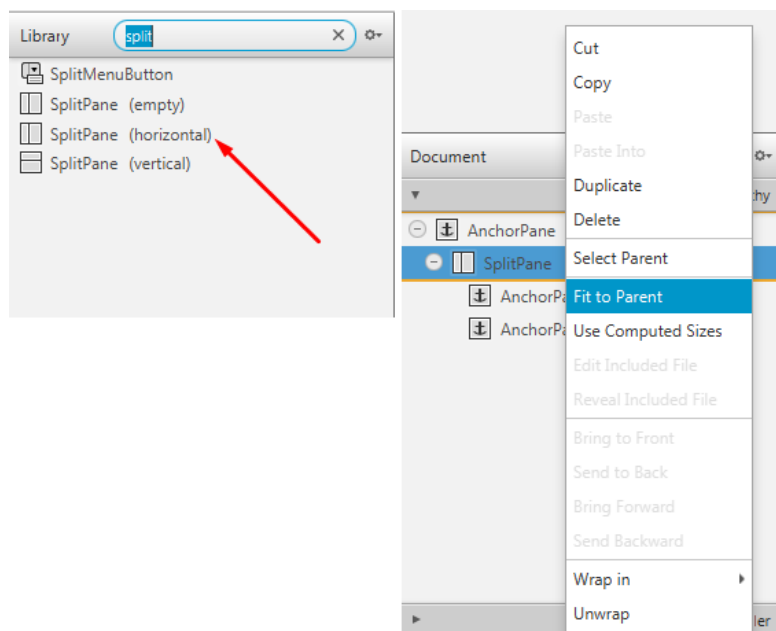


Ya abierto Scene Builder

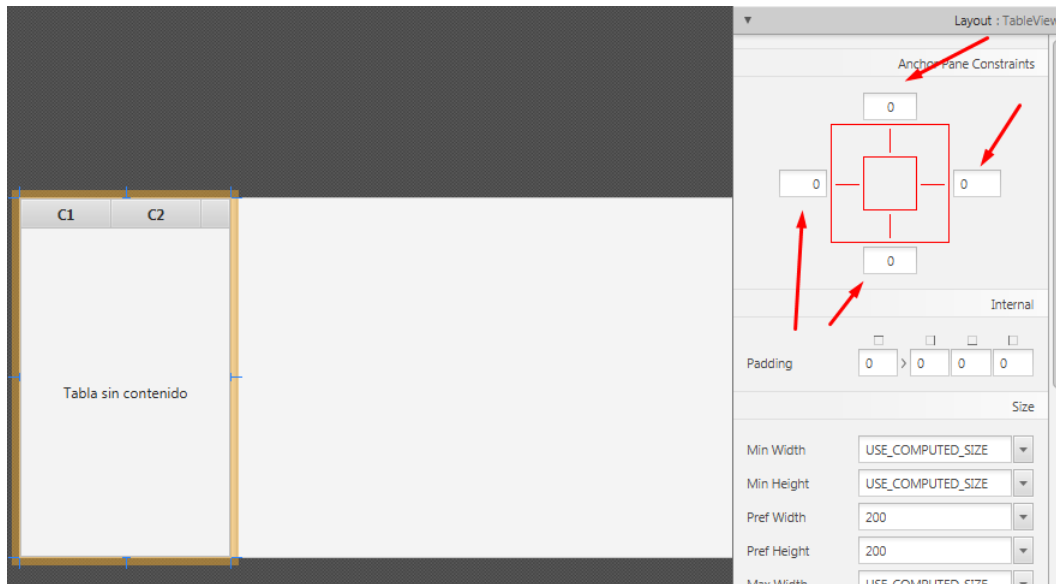
1. Selecciona el `AnchorPane` en tu jerarquía y ajusta el tamaño en el apartado Layout (a la derecha):



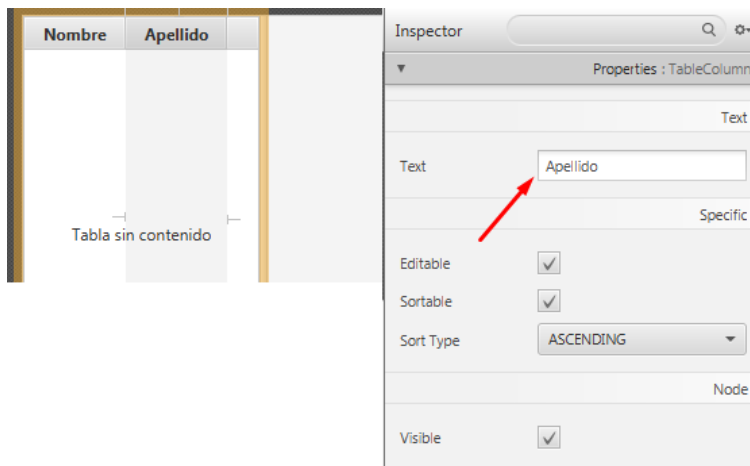
2. Añade un *SplitPane (Horizontal Flow)* arrastrándolo desde la librería (Library) al área principal de edición. Haz clic derecho sobre el SplitPane en la jerarquía y elige Fit to Parent.



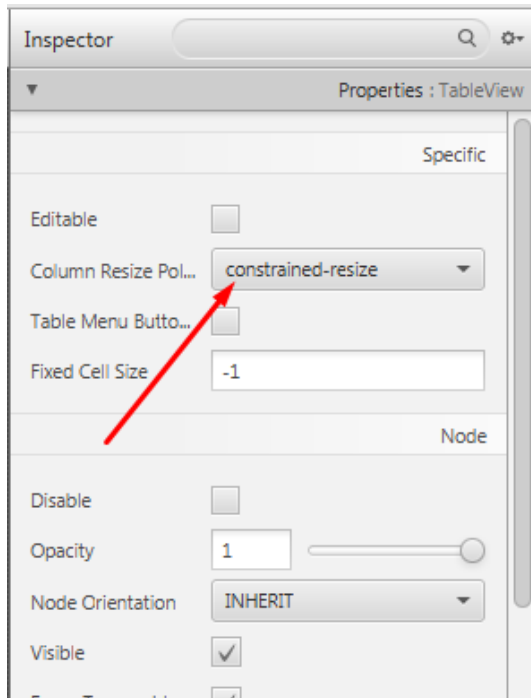
3. Arrastra un TableView (bajo Controls) al lado izquierdo del SplitPane. Selecciona la TableView (no una sola columna) y establece las siguientes restricciones de apariencia (Layout) para la TableView. Dentro de un AnchorPane siempre se pueden establecer anclajes (anchors) para los cuatro bordes ([más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



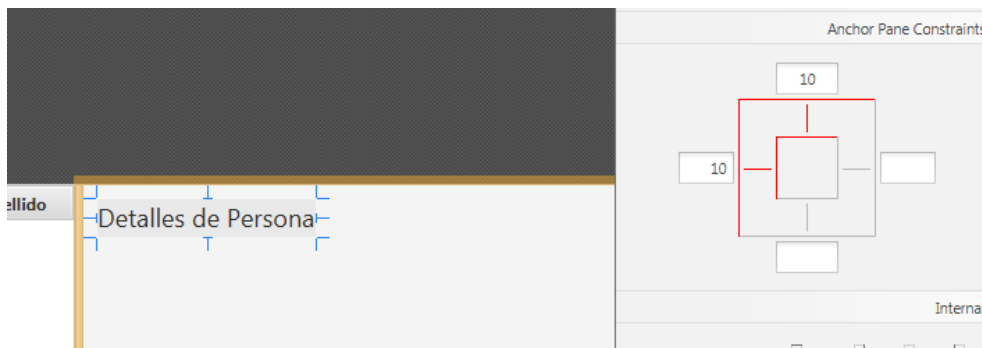
4. Ve al menú *Preview* — *Show Preview in Window* para comprobar si se visualiza correctamente. Intenta cambiar el tamaño de la ventana. La TableView debería ajustar su tamaño al tamaño de la ventana, pues está “anclada” a sus bordes.
5. Cambia el texto de las columnas (en *Properties*) a “Nombre” y “Apellido”.



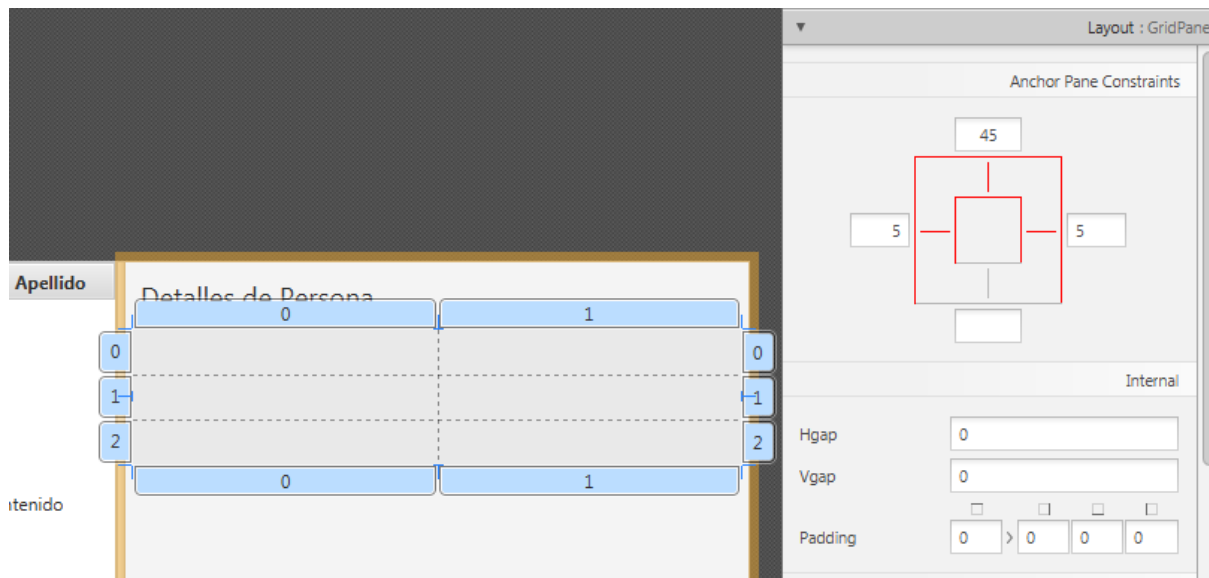
6. Selecciona la *TableView* y elige *constrained-resize* para la *Column Resize Policy* (en *Properties*). Esto asegura que las columnas usarán siempre todo el espacio que tengan disponible.



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.

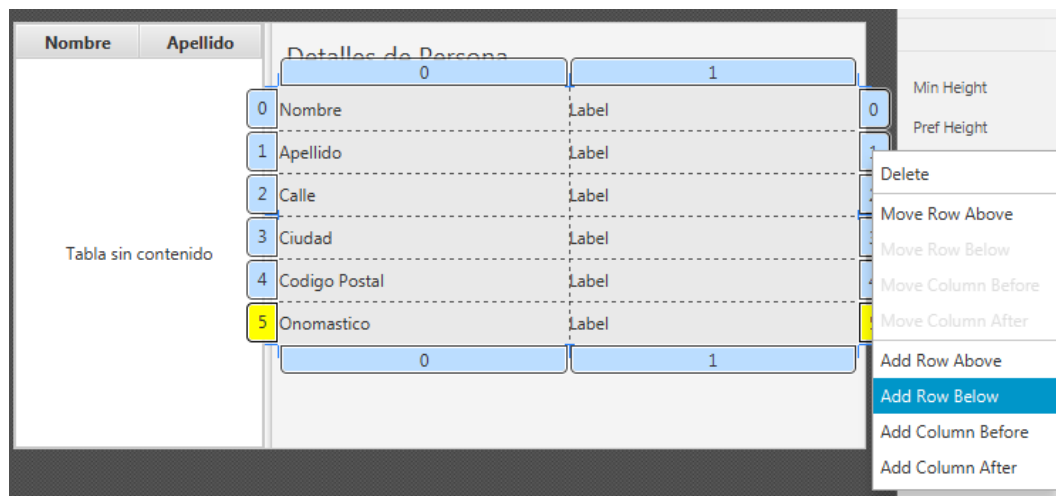


8. Añade un *GridPane* al lado derecho, selecciónalo y ajusta su apariencia usando anclajes (superior, derecho e izquierdo).

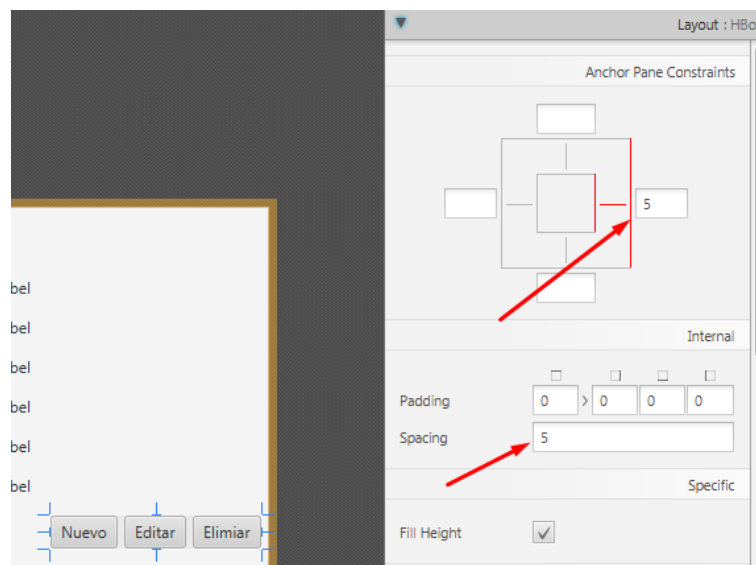
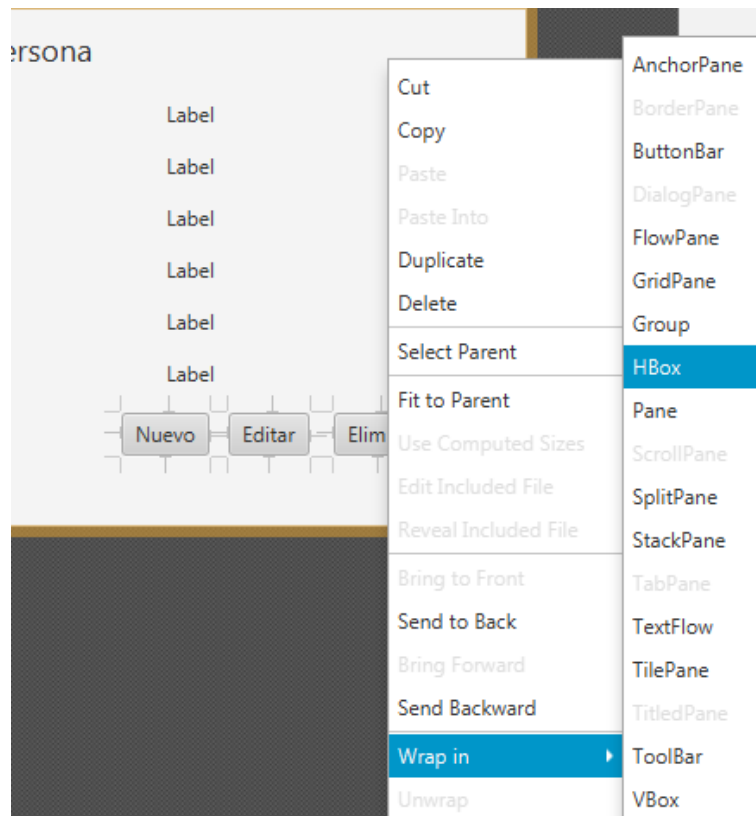


9. Añade las siguientes etiquetas (*Label*) a las celdas del *GridPane*.

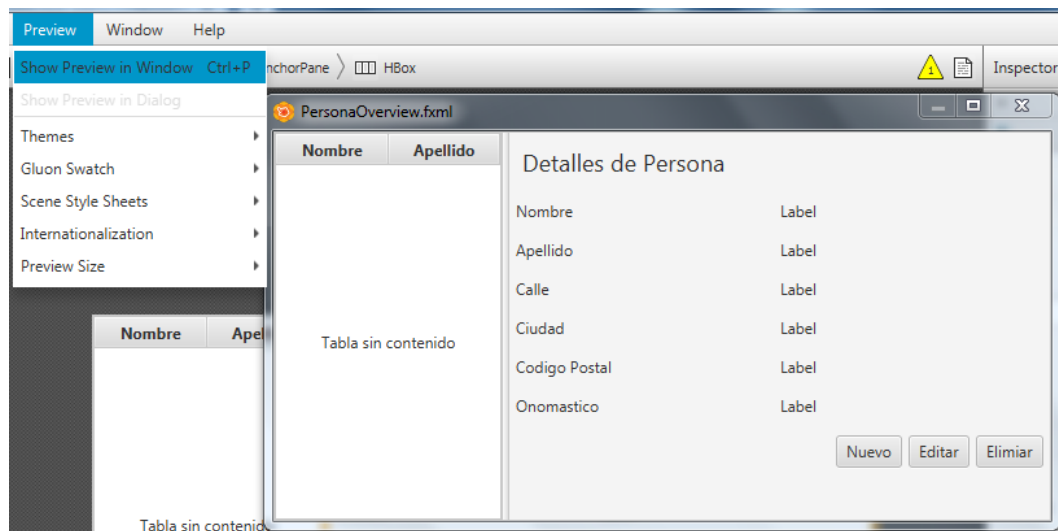
Nota: Para añadir una fila al *GridPane* selecciona un número de fila existente (se volverá de color amarillo), haz clic derecho sobre el número de fila y elige “Add Row”.



10. Añade tres botones a la parte inferior. Truco: Selecciónalos todos, haz clic 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.

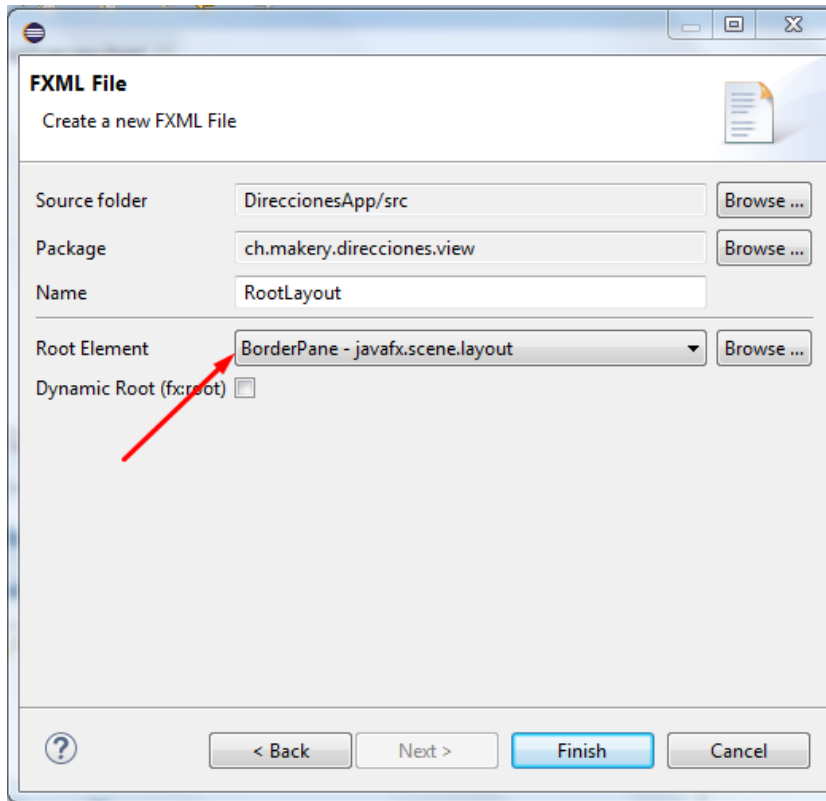


Nota: Guarda los cambios en SceneBuilder, el archivo se actualizará automáticamente en Eclipse, si esto no sucede, puedes presionar F5 para actualizar el archivo.

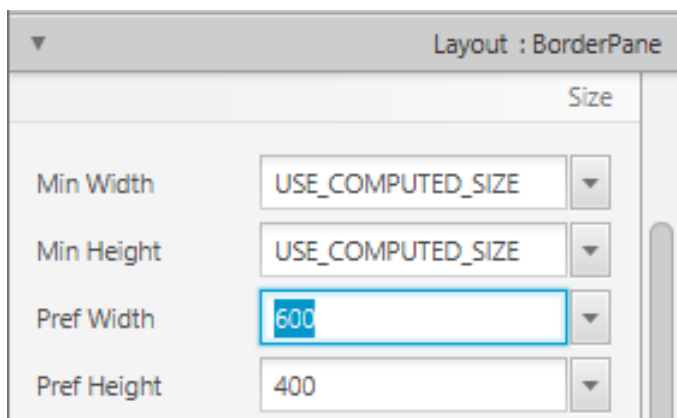
1.8. Crea la aplicación principal

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

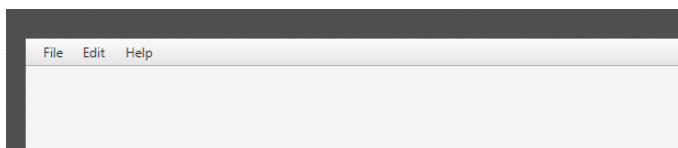
1. Crea otro archivo FXML dentro del paquete view llamado `RootLayout.fxml`. Esta vez, elige `BorderPane` como elemento raíz.



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.



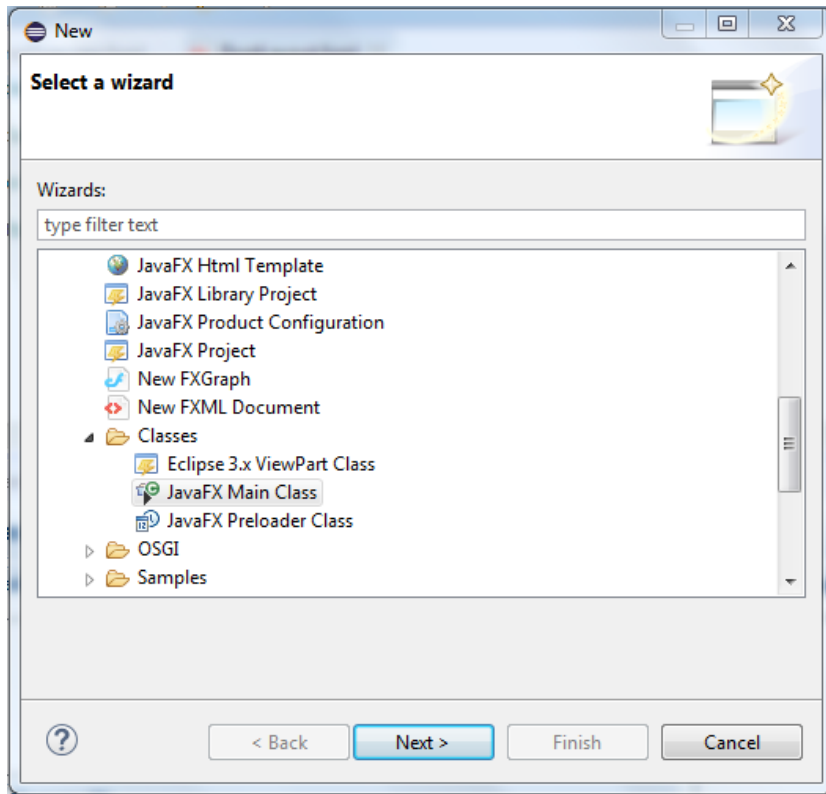
4. Añade una *MenuBar* en la ranura superior del *BorderPane*. De momento no vamos a implementar la funcionalidad del menú.



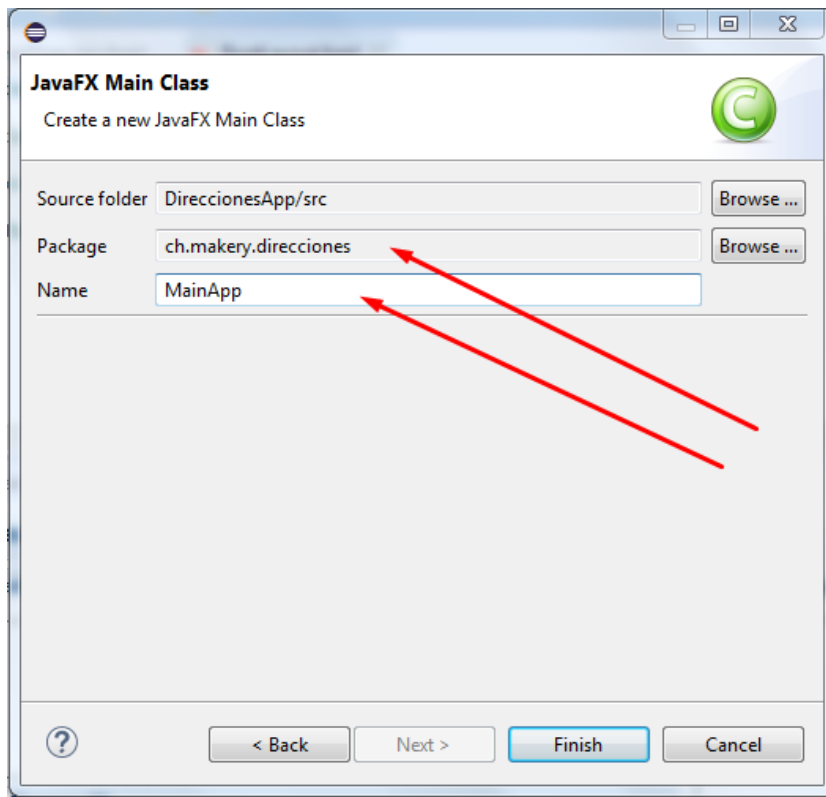
1.9. La clase principal en JavaFX

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

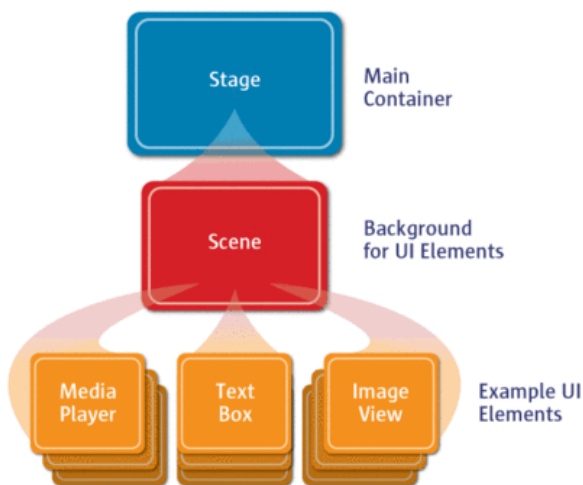
1. Haz clic derecho en el proyecto y elige *New — Other — JavaFX — classes — JavaFX Main Class*.



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(...)` toma 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.fxml"));
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
43      ↪ raíz.
44      */
45     public void showPersonaOverview(){
46         try{
47             //Carga datos de persona
48             FXMLLoader loader = new FXMLLoader();
49             loader.setLocation(MainApp.class.getResource("view/PersonaOvervi
50             ↪ loader.load();
51             //Carga los datos de la persona en el centro del
52             ↪ diseño raíz.
53             rootLayout.setCenter(personaOverview);
54         }catch (IOException e) {
55             e.printStackTrace();
56         }
57     }
58     /**
59      * Retorna el escenario principal
60      * @return
61      */
62     public Stage getPriStage(){
63         return primaryStage;
64     }
65     public static void main(String[] args) {
66         launch(args);
67     }

```

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.

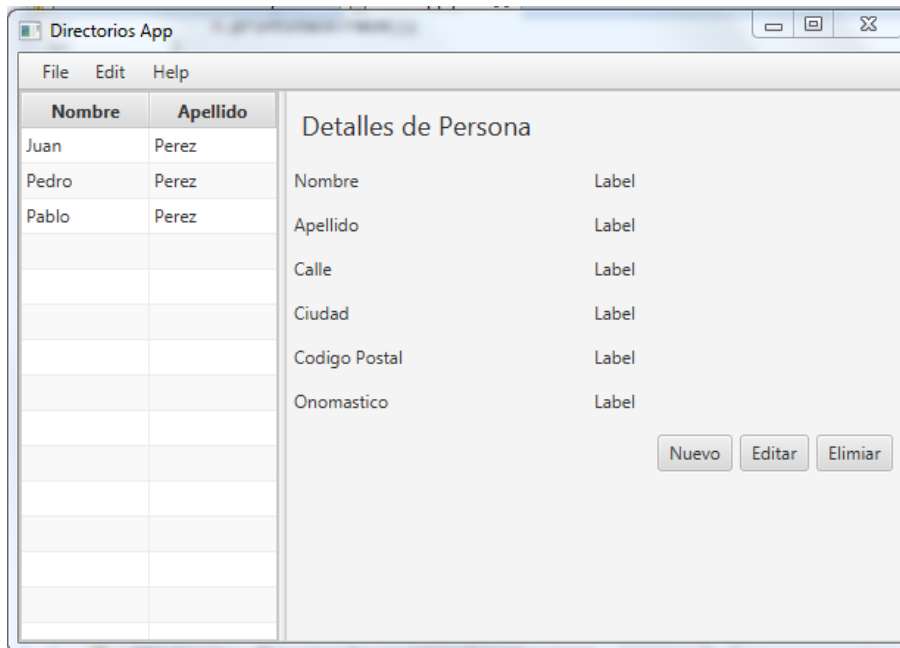
1.10. Problemas frecuentes

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

```
1 package ch.makery.direcciones.model;
2
3 import java.time.LocalDate;
4
5 import javafx.beans.property.IntegerProperty;
6 import javafx.beans.property.ObjectProperty;
7 import javafx.beans.property.SimpleIntegerProperty;
8 import javafx.beans.property.SimpleObjectProperty;
9 import javafx.beans.property.SimpleStringProperty;
10 import javafx.beans.property.StringProperty;
11
12 /**
13  * Modelo de la clase persona
14  * @author Mateo Mtz
15  *
16  */
17 public class Persona {
18     private StringProperty nombre;
19     private StringProperty apellido;
20     private StringProperty calle;
21     private IntegerProperty codigoPostal;
22     private StringProperty ciudad;
23     private ObjectProperty<LocalDate> onomastico;
24     public Persona(String nombre, String apellido){
25         this.nombre = new SimpleStringProperty(nombre);
26         this.apellido = new SimpleStringProperty(apellido);
27
28         //
29         this.calle = new SimpleStringProperty("Alguna calle");
30         this.codigoPostal = new SimpleIntegerProperty(1234);
31         this.ciudad = new SimpleStringProperty("Alguna ciudad");
32         this.onomastico = new
33             ↳ SimpleObjectProperty<LocalDate>(LocalDate.of(1999, 2,
34             ↳ 21));
35     }
36     public StringProperty getNombre() {
37         return nombre;
38     }
39     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
57             ↪ SimpleIntegerProperty(codigoPostal);
58     }
59     public StringProperty getCiudad() {
60         return ciudad;
61     }
62     public void setCiudad(String ciudad) {
63         this.ciudad = new SimpleStringProperty(ciudad);
64     }
65     public ObjectProperty<LocalDate> getOnomastico() {
66         return onomastico;
67     }
68     public void setOnomastico(LocalDate onomastico) {
69         this.onomastico = new
70             ↪ SimpleObjectProperty<LocalDate>(onomastico);
71     }

```

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 =
5          ↪ FXCollections.observableArrayList();
6      /**
7       * Constructor
8       */
9      public MainApp(){
10         //Agregar algunas personas a la lista
11         personData.add(new Persona("Juan", "Perez"));
12         personData.add(new Persona("Pedro", "Perez"));
13         personData.add(new Persona("Pablo", "Perez"));
14     }
15     /**
16     * Devuelve los datos como una lista observable de personas.
17     * @return
18     */
19     public ObservableList<Persona> getPersonData(){
20         return personData;
21     }
```

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 apellidoLabel;
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:

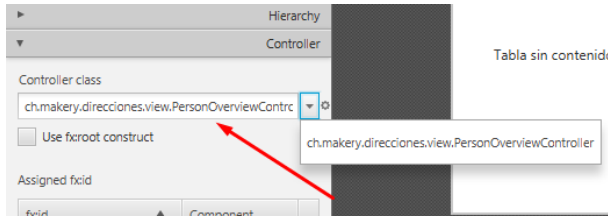
`MainApp.java` - nuevo método `showPersonOverview()`

```
1      /**
2      * Muestra la descripción general de la persona dentro del diseño
↪ raíz.
3      */
4      public void showPersonOverview(){
5          try{
6              //Carga datos de persona
7              FXMLLoader loader = new FXMLLoader();
8              loader.setLocation(MainApp.class.getResource("view/PersonOverview.fxml"));
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 - líneas
↪ adicionales
15              PersonOverviewController controller =
↪ loader.getController();
16              controller.setMainApp(this);
17          }catch (IOException e) {
18              e.printStackTrace();
19          }
20      }
```

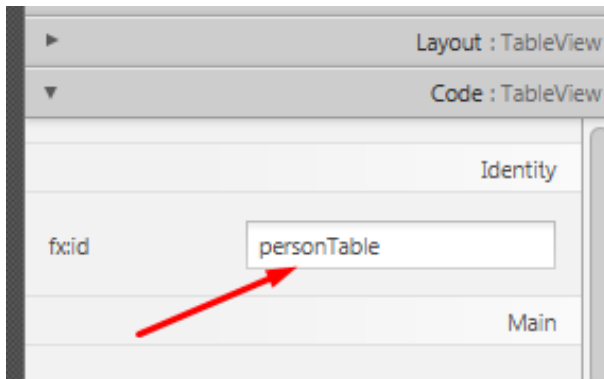
2.5. Vincular la vista al controlador

¡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.

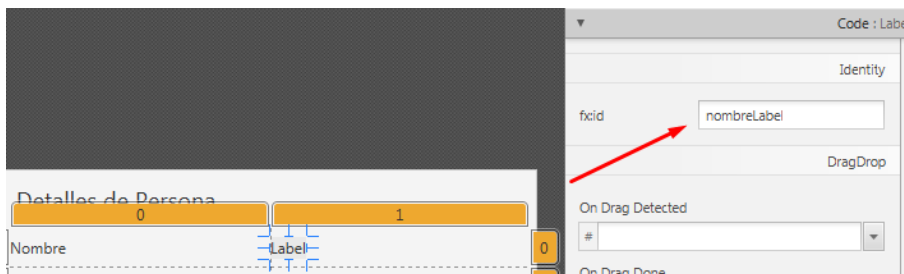
1. `PersonOverview.fxml` en *SceneBuilder*.
2. Abre la sección *Controller* en el lado izquierdo y selecciona `PersonOverviewController` como **controlador**.



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 proyecto 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 `Persona` 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         apellidoLabel.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         apellidoLabel.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
39          ↪ retorna nulo
40          */
41         public static LocalDate parse(String dateString){
42             try {
43                 return DATE_FORMATTER.parse(dateString, LocalDate::from);
44             } catch (Exception e) {
45                 return null;
46             }
47         }
48         /**
49          * Comprueba la cadena si es una fecha valida
50          * @param dateString
51          * @return verdadero si la cadena es una fecha valida.
52          */

```

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