**Parte 3: Interacción con el usuario**

* **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**.

**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 **showPersonDetails(Person person).** 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**

/\*\*

\* Rellenar las etiquetas para ver los datos personales de las personas.

\* Si los campos de una están vacios no mostrará nada

\* @param person the person or null

\*/

private void showPersonDetails(Person person) {

if (person != null) {

// Rellena las etiquetas con la información del objeto Person.

firstNameLabel.setText(person.getFirstName());

lastNameLabel.setText(person.getLastName());

streetLabel.setText(person.getStreet());

postalCodeLabel.setText(Integer.toString(person.getPostalCode()));

cityLabel.setText(person.getCity());

**// TODO: Hay que convertir la fecha de cumpleaños a un string.**

**// birthdayLabel.setText(...);**

} else {

// Si el objeto peson es null dejar todo en blanco.

firstNameLabel.setText("");

lastNameLabel.setText("");

streetLabel.setText("");

postalCodeLabel.setText("");

cityLabel.setText("");

birthdayLabel.setText("");

}

}

**Convierte la fecha de nacimiento en una cadena**

Te darás cuenta de que no podemos usar el atributo birthday directamente para establecer el valor de una Label porque se requiere un String, y birthday es de tipo LocalDate. Así pues necesitamos convertir birthday 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.address.util**:

**DateUtil.java**

package ch.makery.address.util;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.time.format.DateTimeParseException;

/\*\*

\* Funciones de ayuda para manejar fechas.

\*

\*/

public class DateUtil {

/\*\* Patrón de fecha usado para esta conversión. Puedes cambiarlo a tu gusto. \*/

private static final String DATE\_PATTERN = "dd.MM.yyyy";

/\*\* Formato de la fecha. \*/

private static final DateTimeFormatter DATE\_FORMATTER =

DateTimeFormatter.ofPattern(DATE\_PATTERN);

/\*\*

\* Devuelve el valor de la fecha con el formato de string definido en la parte superior.

\* {@link DateUtil#DATE\_PATTERN} es el utilizado.

\* @ el parámetro fecha lo convertirá a tipo string

\* @ lo devuelve en formato string

\*/

public static String format(LocalDate date) {

if (date == null) {

return null;

}

return DATE\_FORMATTER.format(date);

}

/\*\*

\* Corvertir un string al formato definido {@link DateUtil#DATE\_PATTERN}

\* a un objeto fecha {@link LocalDate}.

\*

\* Devolverá null si el string no puede ser convertido.

\*

\* @param dateString la fecha con formato string

\* @devuelve el objeto fecha o nulo si o puede convertirlo

\*/

public static LocalDate parse(String dateString) {

try {

return DATE\_FORMATTER.parse(dateString, LocalDate::from);

} catch (DateTimeParseException e) {

return null;

}

}

/\*\*

\* Chequeo de si el string es una fecha válida.

\*

\* @parametro dateString

\* @devuelve true si el string es una fecha válida

\*/

public static boolean validDate(String dateString) {

// Try to parse the String.

return DateUtil.parse(dateString) != null;

}

}

**Nota:** Puedes cambiar el formato de la fecha cambiando el patrón DATE\_PATTERN. Para conocer los diferentes tipos de formato consulta **[DateTimeFormatter](http://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html)**.

<http://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

**Utilización de la clase DateUtil**

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

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

**Detecta cambios de selección en la tabla**

Para enterarse de que el usuario ha seleccionado a una 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](http://docs.oracle.com/javase/8/javafx/api/) 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 **PersonOverviewController**. El código resultante se asemejará al siguiente:

**PersonOverviewController.java**

@FXML

private void initialize() {

// Inicializa la tabla persona con dos columnas.

firstNameColumn.setCellValueFactory(

cellData -> cellData.getValue().firstNameProperty());

lastNameColumn.setCellValueFactory(

cellData -> cellData.getValue().lastNameProperty());

// Borrar los detalles de persona.

showPersonDetails(null);

// Escucha si cambia la selección y muestra los detalles de la persona cuando cambia.

personTable.getSelectionModel().selectedItemProperty().addListener(

(observable, oldValue, newValue) -> showPersonDetails(newValue));

}

Con showPersonDetails(null); borramos los detalles de una persona.

Con personTable.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 table, nuestra *lambda expression* será ejecutada: se toma la persona recien seleccionada y se le pasa al método showPersonDetails(...) method.

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 PersonOverviewController con [PersonOverviewController.java](http://code.makery.ch/assets/library/javafx-8-tutorial/part3/PersonOverviewController.java).

**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 PersonOverviewController:

**PersonOverviewController.java**

/\*\*

\* Ejecutar cuando el usuario hace click en el botón borrar.

\*/

@FXML

private void handleDeletePerson() {

int selectedIndex = personTable.getSelectionModel().getSelectedIndex();

personTable.getItems().remove(selectedIndex);

}

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

**Gestión de errores**

¿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).

Vamos a añadir un diálogo emergente para informar al usuario. Desafortunadamente no hay componentes para diálogos incluidos en JavaFX 8. Para evitar tener que crearlos manualmente podemos **añadir una librería** que ya los incluya ([Dialogs](http://code.makery.ch/blog/javafx-8-dialogs/)):

1. Descarga este [controlsfx-8.0.6\_20.jar](https://github.com/marcojakob/tutorial-javafx-8/releases/download/v1.0/controlsfx-8.0.6_20.jar) (también se puede obtener de la [página web de ControlsFX](http://fxexperience.com/controlsfx/)). http://fxexperience.com/controlsfx/

**Importante: La versión de ControlsFX debe ser la 8.0.6\_20 o superior para que funcione con JDK 8u20 debido a un cambio crítico en esa versión.**

1. Ve a la subcarpeta **lib** dentro del proyecto y coloca dentro del archivo jar.
2. Añade la librería al **classpath** de tu proyecto: Botón dececho ratón en lib y “Add jar/forder”.

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.java**

/\*\*

\* Ejecutar cuando el usuario hace click en el botón borrar.

\*/

@FXML

private void handleDeletePerson() {

int selectedIndex = personTable.getSelectionModel().getSelectedIndex();

if (selectedIndex >= 0) {

personTable.getItems().remove(selectedIndex);

} else {

// Nothing selected.

Dialogs.create()

.title("Sin selección")

.masthead("No hay ninguna persona seleccionada.")

.message("Seleccione una persona en la tabla.")

.showWarning();

**// ventana de sistema de swing**

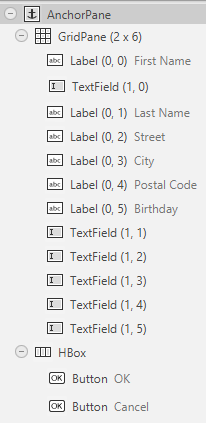
// javax.swing.JOptionPane.showMessageDialog(null, "Error, no has seleccionado a nadie");

}

}

**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.

**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:

**Crea el controlador para la ventana.**

Crea el controlador para la ventana de edición de personas y llámalo

 PersonEditDialogController.java:

**PersonEditDialogController.java**

package ch.makery.address.view;

import javafx.fxml.FXML;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import org.controlsfx.dialog.Dialogs;

import ch.makery.address.model.Person;

import ch.makery.address.util.DateUtil;

/\*\*

\* Ventana de diálogo para editar los detalles de una persona.

\*

\*/

public class PersonEditDialogController {

@FXML

private TextField firstNameField;

@FXML

private TextField lastNameField;

@FXML

private TextField streetField;

@FXML

private TextField postalCodeField;

@FXML

private TextField cityField;

@FXML

private TextField birthdayField;

private Stage dialogStage;

private Person person;

private boolean okClicked = false;

/\*\*

\* Inicializa el controlador de clase. Este método es llamado automáticamente

\* después de que el fichero fxml haya sido cargado.

\*/

@FXML

private void initialize() {

}

/\*\*

\* Carga la ventana (escenario) de diálogo.

\* @param dialogStage

\*/

public void setDialogStage(Stage dialogStage) {

this.dialogStage = dialogStage;

}

/\*\*

\* Carga la clase persona para que sea editada.

\*

\* @param person

\*/

public void setPerson(Person person) {

this.person = person;

firstNameField.setText(person.getFirstName());

lastNameField.setText(person.getLastName());

streetField.setText(person.getStreet());

postalCodeField.setText(Integer.toString(person.getPostalCode()));

cityField.setText(person.getCity());

birthdayField.setText(DateUtil.format(person.getBirthday()));

birthdayField.setPromptText("dd.mm.yyyy");

}

/\*\*

\* Devuelve true si el usuario ha pulsado OK, false si no lo ha pulsado.

\* @return

\*/

public boolean isOkClicked() {

return okClicked;

}

/\*\*

\* Lo ejecuta cuando el usuario pulsa ok.

\*/

@FXML

private void handleOk() {

if (isInputValid()) {

person.setFirstName(firstNameField.getText());

person.setLastName(lastNameField.getText());

person.setStreet(streetField.getText());

person.setPostalCode(Integer.parseInt(postalCodeField.getText()));

person.setCity(cityField.getText());

person.setBirthday(DateUtil.parse(birthdayField.getText()));

okClicked = true;

dialogStage.close();

}

}

/\*\*

\* Lo ejecuta cuando el usuario pulsa cancel.

\*/

@FXML

private void handleCancel() {

dialogStage.close();

}

/\*\*

\* Valida la nueva entrada de usuario en los campos de texto.

\*

\* @ Devuelve true si la entrada es válida

\*/

private boolean isInputValid() {

String errorMessage = "";

if (firstNameField.getText() == null || firstNameField.getText().length() == 0) {

errorMessage += "Nombre no válido!\n";

}

if (lastNameField.getText() == null || lastNameField.getText().length() == 0) {

errorMessage += "Apellido no válido!\n";

}

if (streetField.getText() == null || streetField.getText().length() == 0) {

errorMessage += "Calle no válida!\n";

}

if (postalCodeField.getText() == null || postalCodeField.getText().length() == 0) {

errorMessage += "Código Postal no válido!\n";

} else {

// try to parse the postal code into an int.

try {

Integer.parseInt(postalCodeField.getText());

} catch (NumberFormatException e) {

errorMessage += "Código Postal no válido (Debe ser un número entero)!\n";

}

}

if (cityField.getText() == null || cityField.getText().length() == 0) {

errorMessage += "Ciudad no válida!\n";

}

if (birthdayField.getText() == null || birthdayField.getText().length() == 0) {

errorMessage += "Fecha de cumpleaños no válida!\n";

} else {

if (!DateUtil.validDate(birthdayField.getText())) {

errorMessage += "Fecha de cumpleaños no válida. Debe tener el formato dd.mm.yyyy!\n";

}

}

if (errorMessage.length() == 0) {

return true;

} else {

// Show the error message.

Dialogs.create()

.title("Datos erroneos")

.masthead("Corrija los datos erroneos")

.message(errorMessage)

.showError();

return false;

}

}

}

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 boolean okClicked se utiliza para determinar si el usuario ha pulsado el botón OK o el botón Cancel.

**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.

**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**

import ch.makery.address.view.PersonEditDialogController;

/\*\*

\* Abre la ventana de diálogo para editar los detalles de una persona específica.

\* Si el usuario pulsa ok, los cambios son guardos en el objeto persona proveido y es

\* devuelto el valor true.

\*/

public boolean showPersonEditDialog(Person person) {

try {

// Carga el fichero fxml y creo un nuevo escenario con la ventana emergente.

FXMLLoader loader = new FXMLLoader();

loader.setLocation(MainApp.class.getResource("view/PersonEditDialog.fxml"));

AnchorPane page = (AnchorPane) loader.load();

// Crea el escenario de diálogo.

Stage dialogStage = new Stage();

dialogStage.setTitle("Edit Person");

dialogStage.initModality(Modality.WINDOW\_MODAL);

dialogStage.initOwner(primaryStage);

Scene scene = new Scene(page);

dialogStage.setScene(scene);

// Añade la clase persona al controlador.

PersonEditDialogController controller = loader.getController();

controller.setDialogStage(dialogStage);

controller.setPerson(person);

//Muestra el diálogo y espera a que el usuario la 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 *new* o *edit*.

**PersonOverviewController.java**

/\*\*

\* Lo ejecuta cuando el usuario pulsa el botón nuevo. Abre la ventana de edición

\* que muestra los detalles de la clase persona.

\*/

@FXML

private void handleNewPerson() {

Person tempPerson = new Person();

boolean okClicked = mainApp.showPersonEditDialog(tempPerson);

if (okClicked) {

mainApp.getPersonData().add(tempPerson);

}

}

/\*\*

\* Lo ejecuta cuando el usuario pulsa el botón editar. Abre la ventana de edición.

\* que muestra los detalles de la clase persona.

\*/

@FXML

private void handleEditPerson() {

Person selectedPerson = personTable.getSelectionModel().getSelectedItem();

if (selectedPerson != null) {

boolean okClicked = mainApp.showPersonEditDialog(selectedPerson);

if (okClicked) {

showPersonDetails(selectedPerson);

}

} else {

// Nothing selected.

Dialogs.create()

.title("Sin Selección")

.masthead("Persona no seleccionada")

.message("Seleccione una persona de la tabla.")

.showWarning();

}

}

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

**¡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.