Diálogos en JavaFX

El objetivo de este documento es explicar el funcionamiento los diálogos emergentes en JavaFX que se dividen en tres categorías: Alert, TextDialog y ChoiceDialog

1. Mostrar y ocultar diálogos, modalidad y resultado

En este apartado veremos los aspectos comunes para mostrar un diálogo y manipular su resultado a través de código Java.

1.1. Mostrar y ocultar

Un diálogo emerge desde el código Java normalmente como resultado de cualquier evento asociado a elementos de la aplicación. Por ejemplo, al hacer click en un botón a través del método **`setOnAction`** se visualiza el diálogo denominado infoAlert.

```
infoButton.setOnAction(e -> {
    infoAlert.showAndWait();
});
```

Para mostrar un diálogo tenemos los métodos **`show`** o **`showAndWait`**. La diferencia reside en que **`showAndWait`** bloquea la ejecución del resto del código hasta que el usuario lleva a cabo alguna acción del diálogo. Además, permite recoger la respuesta del usuario mediante un objeto de la clase **Optional**.

Para cerrar un diálogo desde el código Java tendremos que ejecutar el método **`close`**.

1.2. Obtener resultado

Como se decía anteriormente, el método **`showAndWait`** permite recoger la respuesta del usuario mediante un objeto de la clase **Optional**.

Es posible obtener este objeto de varias maneras. Por ejemplo, en un diálogo de tipo **Confirmation** tenemos las siguientes posibilidades:

- Enfoque clásico: Se recoge el resultado del método **showAndWait** en una variable. Se comprueba con el método **isPresent** si el usuario ha presionado algún botón y en caso afirmativo se obtiene el tipo de botón seleccionado **get**.

```
Optional<ButtonType> result = confirm.showAndWait();
if (result.isPresent() && result.get() == ButtonType.OK) {
   textInput.setText("");
   textInput.requestFocus();
}
```

- Enfoque clásico con función lambda: El método **ifPresent** permite directamente ejecutar código si el usuario ha seleccionado el botón "Aceptar".

```
confirm.showAndWait().ifPresent(response -> {
   if (response == ButtonType.OK) {
      textInput.setText("");
      textInput.requestFocus();
   }
});
```

- Enfoque solamente con función lambda: Permite filtrar previamente el tipo de respuesta del botón "Aceptar" y luego ejecutar el código asociado a dicha acción solo si el usuario la ha llevado a cabo.

```
confirm.showAndWait()
   .filter(response -> response == ButtonType.OK)
   .ifPresent(response -> {
      textInput.setText("");
      textInput.requestFocus();
   });
```

Más adelante, veremos el funcionamiento de estos métodos según los diferentes tipos de diálogo.

1.3. Modalidad

Un diálogo puede ser:

- **Modal:** Requieren que el usuario responda antes de continuar con el programa.
- **No modal:** que permanecen en la pantalla y están disponible para su uso en cualquier momento, pero permiten otras actividades de usuario.

Para ello, las clases que heredan de **Dialog** (o Stage en el caso de diálogos personalizados que veremos más adelante) tienen un método denominado **initModality** que admite tres valores que consisten en propiedades estáticas de la clase Modality:

- **Modality.NONE:** Permite crear diálogos no modales y por tanto se puede hacer click en otra sección de la aplicación.
- **Modaliy.APPLICATION_MODAL:** Permite bloquear eventos de cualquier otra ventana de la aplicación. Es decir, el diálogo se queda fijo y no se puede acceder al resto de la aplicación hasta que se cierra.

- **Modaliy.WINDOW_MODAL:** Permite bloquear eventos de la jerarquía de ventanas a la que pertenece. Para ello, tenemos que especificar el objeto **Stage** de la ventana que genera el diálogo a través del método **setOwner** como se indica debajo. En la práctica es muy parecido a la opción anterior y solamente en aplicaciones complejas con muchas ventanas tiene un sentido práctico bloquear solo una serie de ventanas específicas.

```
dialogStage.initModality(Modality.WINDOW_MODAL);
dialogStage.initOwner(primaryStage);
```

Los diálogos de las clases **Alert**, **TextDialog**, y **ChoiceDialog** que estudiaremos más adelante son modales por defecto. Para cambiar a "no modal" necesitaríamos el valor **NONE** correspondiente. Por ejemplo:

```
choiceDialog.initModality(Modality.NONE);
```

2. Alert

Como su propio nombre indica, permite mostrar al usuario alertas de diferentes tipos: confirmación, error, información, advertencia y genérico.

Un diálogo es una ventana emergente que normalmente se activa como respuesta a otra acción, por ejemplo, los botones de la aplicación de debajo que tomaremos como referencia.

Al hacer click en el botón "Info" de la ventana se generará un diálogo de tipo informativo. En principio no tenemos la posibilidad de configurar este elemento desde **SceneBuilder** y tendremos que codificarlo desde Java con la clase correspondiente **Alert.**

Previamente creamos un diálogo con la clase **Alert** indicando en el constructor que es del tipo informativo.

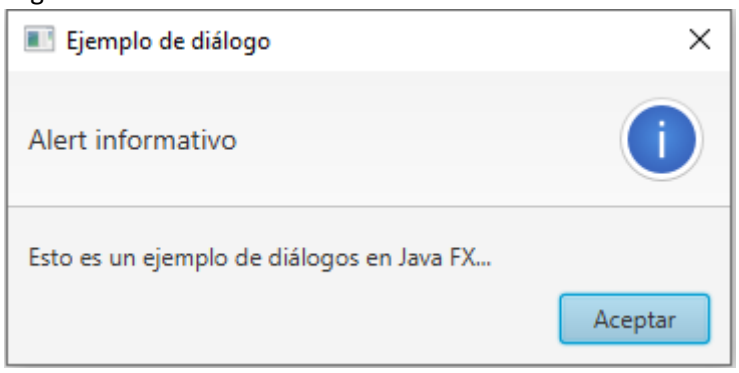
```
Alert infoAlert = new Alert(AlertType.INFORMATION);

infoAlert.setTitle("Ejemplo de diálogo");
infoAlert.setHeaderText("Alert informativo");
infoAlert.setContentText("Esto es un ejemplo de diálogos en Java FX...");
```

Al hacer click en un elemento de tipo **Button** se desplegará el diálogo creado anteriormente.

```
infoButton.setOnAction(e -> {
    infoAlert.showAndWait();
});
```

El resultado es el siguiente:

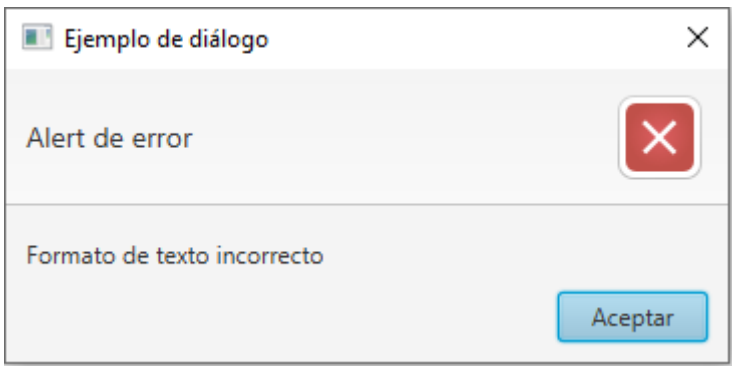


Teniendo en cuenta el código de arriba para crear el diálogo, podemos observar lo siguiente:

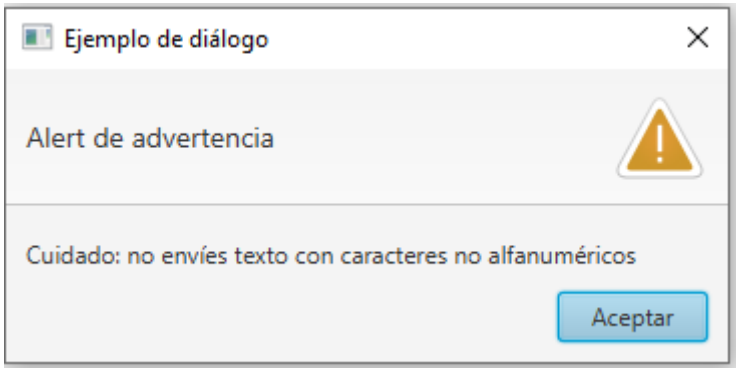
- El texto de la ventana "Ejemplo de diálogo" se corresponde al método **`setTitle`**.
- El texto de la cabecera "Alert informativo" se corresponde al método **`setHeaderText`**.
- El texto que aparece encima del botón "Aceptar" es una descripción del contenido del diálogo y se corresponde con el método **`setContextText.`**
- El icono es de tipo informativo, pero veremos más adelante que se puede modificar según el tipo de diálogo. En este caso era `AlertType.`***`INFORMATION.`***
- El diálogo se muestra con **`showAndWait`** como se explicaba en el primer apartado del documento.

Los diálogos del resto de tipos se crean de la misma forma cambiando el valor del tipo **AlertType** del constructor por el que corresponda en cada caso y como se indica a continuación:

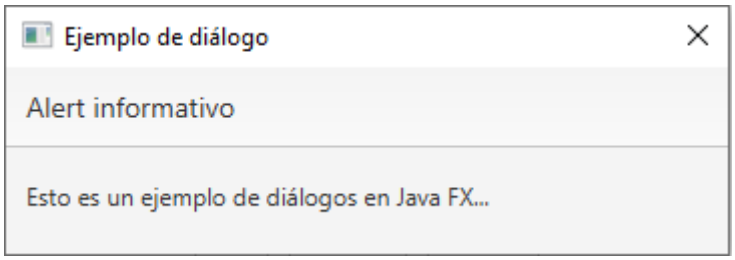
- Diálogo de error `AlertType.`***`ERROR:`***



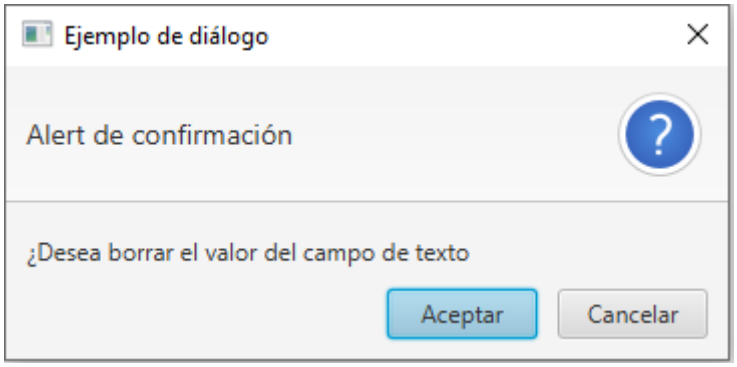
- Diálogo de advertencia `AlertType.`***`WARNING:`***



- Diálogo genérico `AlertType.NONE:`



- Diálogo de confirmación `AlertType.CONFIRMATION:`



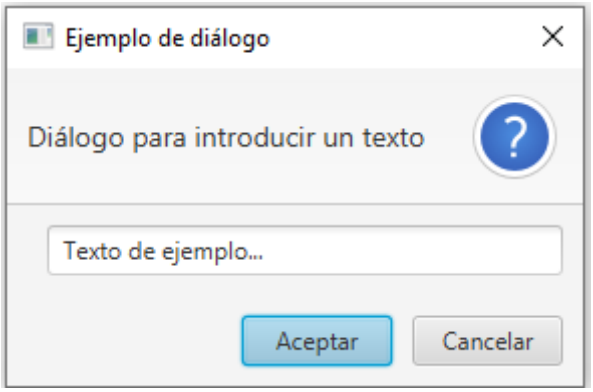
Respecto al último tipo de "Confirmación", es posible obtener la respuesta del usuario con el objeto **Optional** como se indicaba anteriormente. En este documento nos centraremos en el enfoque tradicional con función lambda, pero se puede obtener el resultado con cualquiera de los métodos mencionados en el primer apartado.

```
confirm.showAndWait().ifPresent(response -> {
   if (response == ButtonType.OK) {
      textInput.setText("");
      textInput.requestFocus();
   }
});
```

En el código anterior, si el usuario hace click en Aceptar (tipo de botón OK), entonces se vacía el campo de texto de la imagen del inicio de la sección actual. Además, se activa el foco sobre este elemento de tipo **TextField**.

3. TextDialog

Se trata de un tipo de diálogo que permite al usuario introducir texto en un campo **TextField** integrado en la ventana emergente. El tipo de diálogo es similar a la imagen siguiente:



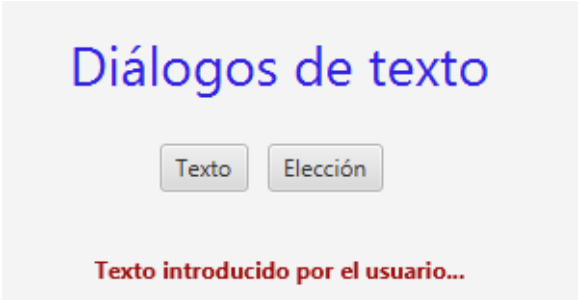
El código que muestra el diálogo anterior es el siguiente, como se puede observar a través de la respuesta al hacer click en un botón.

```
// Diálogo con texto por defecto
TextInputDialog textDialog = new TextInputDialog("Texto de ejemplo...");
textDialog.setTitle("Ejemplo de diálogo");
textDialog.setHeaderText("Diálogo para introducir un texto");

textButton.setOnAction(e -> {
	// Se incluye la respuesta del usuario un campo de tipo Text
	textDialog.showAndWait().ifPresent(response -> {
		userInput.setText(response);
	});
});
```

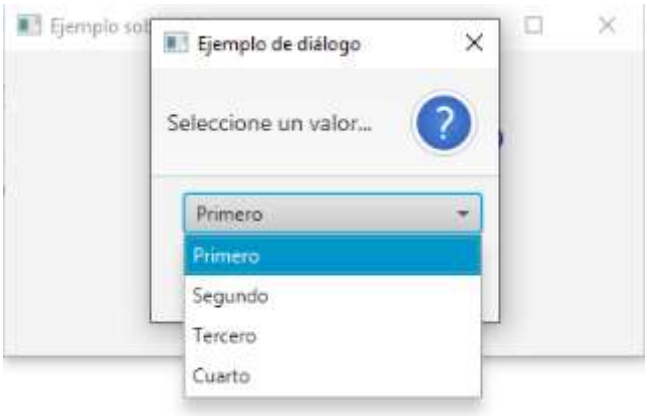
Del código de arriba cabe detallar lo siguiente:

- El "Texto de ejemplo…" que se indica como parámetro en el constructor es el que aparece por defecto en el campo de texto del diálogo.
- El texto de la ventana "Ejemplo de diálogo" se corresponde al método **`setTitle`**.
- El texto de la cabecera "Diálogo para introducir un texto" se corresponde al método **`setHeaderText`**.
- El texto introducido por el usuario se guarda en el elemento userInput de tipo **Text** y se muestra de una manera similar a la ventana de debajo (se corresponde con el texto rojo).



4. ChoiceDialog

Se trata de un diálogo que muestra un elemento similar a un **ComboBox** en el que el usuario puede elegir uno de los valores. A continuación se muestra un ejemplo:



Para crear este tipo de diálogo, tendremos que crear previamente la lista de valores (en este caso de tipo String) y pasársela como parámetro al constructor **ChoiceDialog**.

```
// Lista para el ChoiceDialog
String [] arrayData = {"Primero", "Segundo", "Tercero", "Cuarto"};
List<String> dialogData = Arrays.asList(arrayData);

// Se crea el ChoiceDialog con el primer elemento seleccionado
ChoiceDialog<String> choiceDialog = new
      ChoiceDialog<String>(dialogData.get(0), dialogData);
choiceDialog.setTitle("Ejemplo de diálogo");
choiceDialog.setHeaderText("Seleccione un valor...");

choiceButton.setOnAction(e -> {
      // Se incluye la respuesta del usuario en un campo de tipo
      choiceDialog.showAndWait().ifPresent(response -> {
            userInput.setText(response);
      });;
});
```

Cabe explicar con más detalle las siguientes líneas del código anterior:

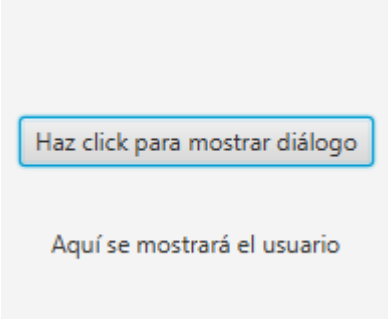
- Los métodos **`setTitle`** y **`setHeaderText`** tienen el mismo significado que en el resto de diálogos estudiados anteriormente. Es decir, definen el texto de la ventana y de la cabecera de la ventana.
- El constructor recibe dos parámetros: el valor que se selecciona por defecto (el primero del listado) y el listado de valores que se van a mostrar en el **ComboBox.**
- El texto introducido se guarda en el elemento userInput de tipo **Text** y se mostrará de la misma manera que en el apartado anterior.

5. Diálogos personalizados

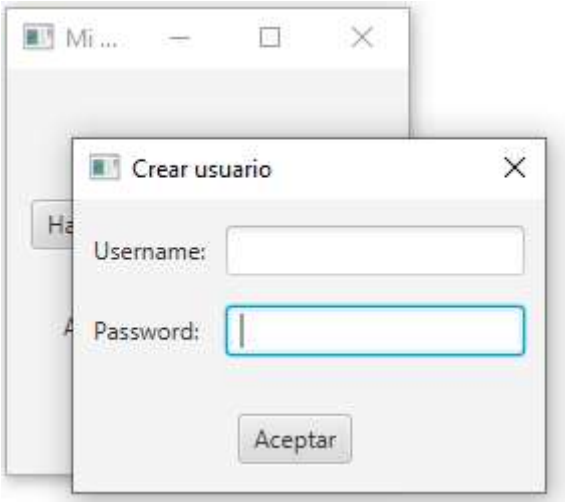
Al margen de los objetos **Alert**, **TextInputDialog** y **ChoiceDialog** que nos proporcionan diálogos, podemos crear nuestras propias ventanas.

Existe la posibilidad de crear clases que extiendan de **Dialog** y modifiquen el diseño. O incluso más sencillo, podemos crear un diseño ya sea con Java o FXML y abrir la ventana como emergente.

Por ejemplo, supongamos la siguiente página:



Al hacer click en el botón se mostrará el diálogo de la imagen de debajo:



Para ello, creamos nuestro diseño en un archivo FXML. Para que se muestre al hacer click en el botón a través del método `setOnAction`, podemos incluir el código que carga el diseño y lo abre como una ventana desplegable.

```
btnMostrar.setOnAction(ev -> {
   try {
      // Cargamos el diseño del diálogo desde un XML
      FXMLLoader loader = new FXMLLoader();

      loader.setLocation(MainDialogoPersonalizado.class.
       getResource("MiDialogo.fxml"));

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

      // Se crea un nuevo Stage para mostrar el diálogo
      Stage dialogStage = new Stage();
      dialogStage.setTitle("Crear usuario");

      // Se bloquean los eventos de la pantalla principal
      dialogStage.initModality(Modality.WINDOW_MODAL);
      dialogStage.initOwner(primaryStage);

      Scene scene = new Scene(page);
      dialogStage.setScene(scene);
      dialogStage.showAndWait();

  } catch (IOException e) {
     e.printStackTrace();
  }

});
```

Como se puede observar arriba, se carga el FXML de manera similar a la ventana principal y se crea un **Stage.** El objeto **Stage** se crea del tipo modal con **`initModality`** para bloquee la pantalla principal cuyo objeto **Stage** se indica en el método **`initOwner`.**

A continuación se añade un objeto **Scene** y se muestra con **`show`** o **`showAndWait`** como en cualquier aplicación.

Por último, cuando se abren diálogos es bastante común acceder a su controlador e intercambiar objetos con la clase **Main.** Por ejemplo, podemos cargar el controlador y asignar propiedades con set de la siguiente manera:

```
MiDialogoController controller = loader.getController();
controller.setDialogStage(dialogStage);
controller.setLabel(lblUsuario);
```

De esta forma, se puede lograr que en la ventana principal aparezcan datos que se han introducido en el controlador (habría que crear las propiedades y setter correspondientes en el propio controlador).