Ficha 11

Java FX

Java FX es la más reciente biblioteca de desarrollo de interfaces gráficas de usuario disponible para programas Java. Provee mejoras frente a sus antecesoras (AWT y Swing). Entre sus características más destacables está la posibilidad de establecer estilos para la aplicacion mediante hojas de estilos CSS, manejo de eventos simplificado y la opción de desarrollo para sistemas operativos de dispositivos móviles.

1.] Instalación

La librería de JavaFX es parte integral de la biblioteca estándar de Java y no requiere la instalación de ningún software extra tanto para el desarrollo como para la ejecución de programas en computadoras que posean instalado el JRE.

La creación de las ventanas que conformen la interfaz puede ser realizada mediante código fuente, creando objetos que en ejecución posean una representación visual, tales como ventanas, botones, etc., sin embargo esta estrategia suele ser utilizada en casos excepcionales ya que es sumamente laboriosa. La programación de una única ventana con un puñado de controles requiere cientos de líneas de código muy simple de escribir pero sumamente detallado, ya que por cada componente que deba visualizarse en una ventana deben establecerse muchas propiedades de cada objeto. Por ejemplo, para algo tan simple como dibujar un botón en una ventana debe crearse una instancia de una clase y luego asignarle las propiedades que modifiquen su texto, posición, colores, bordes, etc. Por otro lado, utilizando esta estrategia no se puede conocer cómo se ve cada ventana hasta que el programa no es ejecutado.

Naturalmente se prefiere utilizar un diseñador visual donde se pueda agregar y diseñar todo el contenido de los elementos visuales de cada ventana mediante mecanismos de drag & drop y en donde el programador pueda visualizar durante el desarrollo el resultado final sin requerir compilar y ejecutar el programa.

Para las librerías más antiguas (AWT y Swing) cada IDE debía desarrollar un diseñador de ventanas propio. Ello tenía dos desventajas principales, ya que cada creador de IDE debía repetir el esfuerzo en crear su diseñador y por otro lado, las ventanas diseñadas con un IDE no podían ser fácilmente modificadas con otro.

En el caso de JavaFX se siguió una estrategia diferente. Sun desarrolló un único diseñador de ventanas denominado SceneBuilder, de forma tal que cada IDE no requiera ofrecer un diseñador. La figura 1 muestra la ventana principal del diseñador. Además si se desea cambiar de IDE, las ventanas pueden ser modificadas sin inconvenientes. Oracle decidió no continuar con el desarrollo de SceneBuilder por lo que lo entregó a la comunidad open source. Una compañia periódicamente toma la última versión del codigo fuente y crea instaladores para diversos sistemas operativos.

El instalador de SceneBuilder puede ser obtenido de `http://gluonhq.com/products/scene-builder/`. Las últimas versiones liberadas son la 9.0 desarrollada para programas compilados con Java 9, y la 8.4 para programas Java 8.

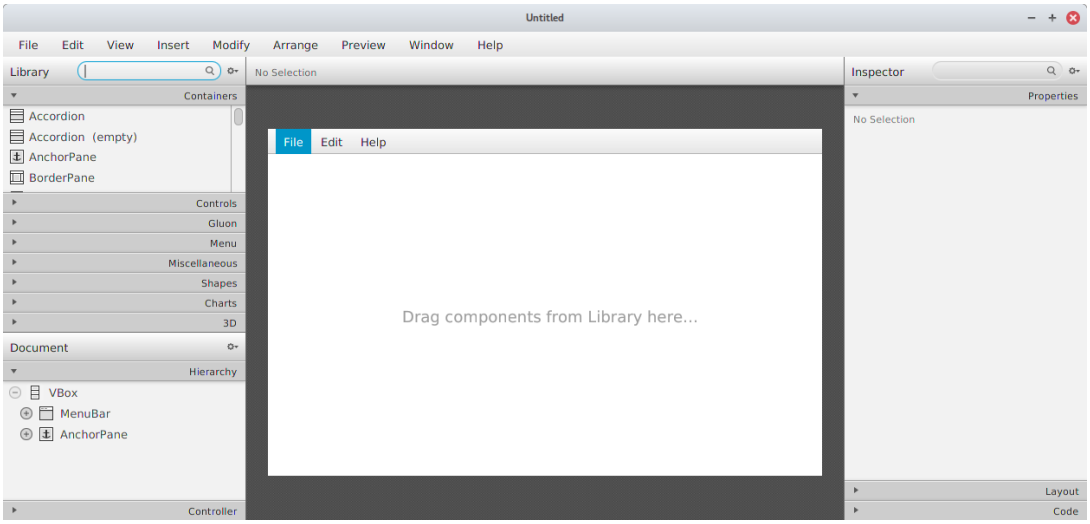

Figura 1: Interfaz principal de SceneBuilder

2.] Estructura de un programa

Los programas JavaFX presentan una particularidad notoria ya que **no poseen método main()**. En cambio, el punto de entrada se especifica mediante una herencia de la clase `Application`.

Por lo tanto para crear un programa JavaFX se debe crear una clase derivada de Application y sobreescribir el método `public void start()`. El método start recibe como parámetro un objeto Stage que representa a la ventana principal. Con ese objeto stage se debe invocar a su método show() para que la ventana se muestre. En ese punto el programa muestra la ventana principal, la cual está inicialmente vacía, y el método start finaliza. El programa continúa la ejecución hasta que se cierren todas las ventanas, tanto la principal como otras que se puedan haber abierto.

El siguiente programa inicia asignando a la ventana principal una cadena que se presenta en la barra de título y finalmente la muestra. La figura 2 muestra la ventana durante la ejecución.

```
import javafx.application.Application;
import javafx.stage.Stage;

public class JavaFXApplication2 extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        stage.setTitle("Primera ventana");
        stage.show();
```

Figura 2: Ventana principal sin contenido

```
    }
}
```

3.] Ventanas

En las interfaces gráficas de usuario para programas de escritorio el objeto principal es la ventana. Las ventanas son los únicos elementos que pueden mostrarse en forma independiente en el escritorio del sistema operativo. Cualquier otro elemento visual (botones, cuadros de texto, imágenes, etc.) debe mostrarse obligatoriamente dentro de una ventana.

Para el armado de una ventana se sigue un concepto que simula una obra de teatro tradicional. En un teatro toda la obra es presentada dentro de un escenario. En este escenario, pueden presentarse diversas escenas una tras la otra en una secuencia, pero no hay dos escenas en forma simultánea. Cuando una escena finaliza, continua la siguiente escena, pudiendo cambiar toda la escenografía.

En esta librería, por cada ventana que se desee mostrar se debe crear un escenario mediante la creación de una instancia de la clase Stage. Luego se debe agregar a dicha stage una instancia de la clase Scene. Durante la ejecución del programa cada ventana (Stage) posee asociada una escena (Scene).

La escena es una superficie rectangular donde se agrega el contenido que se desea mostrar al usuario. Durante la vida de una ventana se puede cambiar su escena por otra, aunque en las aplicaciones de escritorio esto no es tan habitual. Este concepto de escenas que van reemplazándose es central en las aplicaciones móviles, en donde todo el programa suele ser ejecutado en una única ventana, la cual a su vez ocupa toda la pantalla del dispositivo móvil.

4.] Archivos FXML

Para llenar de contenido una escena el mecanismo preferido involucra la creación de archivos xml que especifican la jerarquía de contención de los componentes visuales y las diversas propiedades de los mismos.

Cada componente visual que se desee agregar a una ventana va a estar contenido dentro de unos objetos denominados contenedores, los cuales a su vez también pueden pertenecer a otros contenedores. De esta forma queda determinada una jerarquía en donde un contenedor denominado root engloba a otros contenedores o componentes visuales. Todos los contenedores heredan de la clase Parent.

Cada uno de estos objetos posee atributos que determinan su presentación, tales como color, tipos de letra, tamaño, etc. y su comportamiento. Los componentes que son utilizados para que el usuario interactúe con la aplicación son denominados genéricamente como controles.

Por cada escena de la aplicación debe redactarse un archivo con formato XML y extensión .fxml. Esta redacción es bastante simple y fácil de memorizar, por ejemplo, el siguiente xml define una ventana que contiene un cuadro de texto y un botón. Ambos controles están contenidos dentro de una instancia de la clase VBox (por Vertical Box) que automáticamente ubica cada control contenido uno debajo del otro.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.VBox?>


<VBox prefHeight="108.0" prefWidth="222.0" spacing="10.0"
↪  xmlns:fx="http://javafx.com/fxml/1">
        <children>
                <TextField />
                <Button text="Click Aquí" />
        </children>
        <padding>
                <Insets bottom="20.0" left="20.0" right="20.0"
                ↪  top="20.0" />
        </padding>
</VBox>
```

Precisamente la tarea del diseñador SceneBuilder es la redacción de estos archivos fxml. La figura 3 muestra la interfaz de SceneBuilder al editar la escena correspondiente a este fxml.

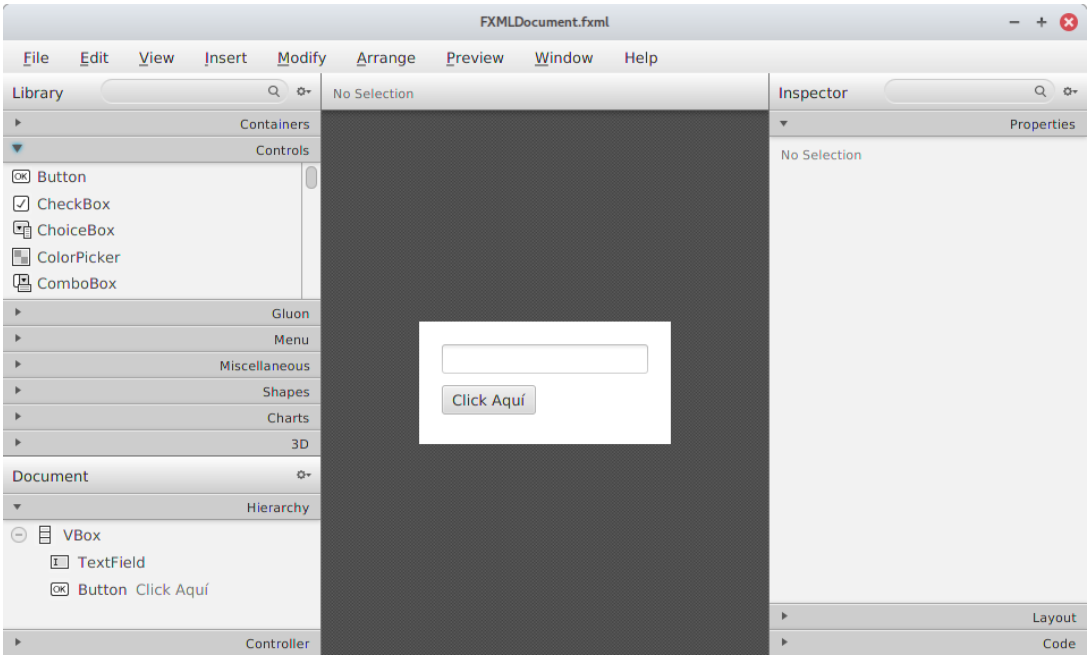

Figura 3: SceneBuilder editando un fxml con un cuadro de texto y un botón

Uso de los archivos fxml desde un programa Java

Los archivos fxml contienen el diseño gráfico de cada escena del programa. para poder utilizarlos se requiere cargarlos en el programa mediante una llamada al método `FXMLLoader.load(url)`, donde el parámetro url indica la ubicación del archivo fxml. Esta ubicación cambia entre el desarrollo y la ejecución, por lo tanto no es recomendable que sea indicada en forma fija en el código. La recomendación principal es la de obtenerla en tiempo de ejecución indicando únicamente el nombre del archivo y permitiendo que la máquina virtual identifique su ubicación dentro del proyecto. Es imprescindible, sin embargo, que el archivo se encuentre incluido en el proyecto, en una carpeta de código fuente o en una carpeta de recursos.

El método `FXMLLoader.load()` retorna una instancia de la clase Parent, es decir una referencia al contenedor principal. En el ejemplo anterior, se retorna una referencia a una instancia de la clase VBox. La carga de la escena por lo tanto se realiza con el siguiente fragmento de código:

```
URL urlEscena1 = getClass().getResource("Escena1.fxml");
Parent root = FXMLLoader.load(urlEscena1);
```

Finalmente para poder hacer visible una ventana con la escena cargada se requiere:

- crear una instancia de `Scene` con el contenedor cargado
- si es la primera ventana del programa, asignar en el stage recibido por parámetro el objeto Scene creado.

- si no es la primera ventana crear una nueva instancia de `Stage` con el Scene creado
- invocar al método `show()` del Stage.

A continuación se presenta un programa completo que muestre la escena del ejemplo anterior. En la figura 4 se presenta la ventana durante la ejecución.

```
package edu.utn.tsb.ejemplo1;

import java.net.URL;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Ejemplo1 extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        URL urlEscena1 = getClass().getResource("Escena1.fxml");
        Parent root = FXMLLoader.load(urlEscena1);

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.setTitle("Primera ventana");
        stage.show();
    }
}
```

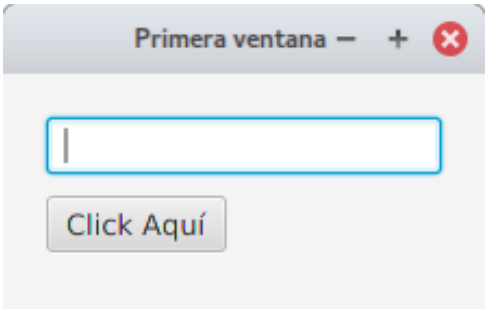

Figura 4: Ejecución del ejemplo 1

5.] Controladores

El comportamiento de las ventanas (especialmente el relacionado con la interacción con el usuario y la detección de las acciones que el usuario realiza con los diversos elementos visuales) debe ser programado en unas clases java que se encontrarán completamente acopladas con el contenido de las escenas. Pero dado que las escenas estan definidas en los archivos fxml es necesario algún mecanismo de emparejamiento entre ambos.

Por cada escena se debe programar una clase denominada controladora, la cual debe implementar la interfaz Initializable. Dado que por cada escena debe existir una clase controladora, la convención es que la misma posea como nombre el mismo que el archivo fxml relacionado, con el sufijo Controller.

Las clases controladoras actúan como puente entre los elementos visuales de la escena y las clases java que implementan la funcionalidad central de la aplicación. Para poder establecer esta relación se requieren código especial que efectúe ese enlace.

Propiedades y controles

Todo elemento de una escena tiene una propiedad denominada “fx:id” donde se almacena el identificador con el que se hará referencia al objeto en el código fuente. El fx:id se establece en el archivo xml, y normalmente lo indica el programador en el SceneBuilder. Este identificador debe cumplir con las reglas sintácticas y convenciones de los nombres de atributos de java.

Por su lado, la clase controladora debe poseer atributos privados cuya definición coincida en nombre y tipo con los indicados en el fxml relacionado. Estos atributos además deben estar marcados con la annotation `@FXML`.

Es muy destacable que no todos los elementos visuales de una escena requieren tener definido su id, sino sólo aquellos que deban ser utilizados por el código del programa. Por ejemplo, en una ventana donde haya un cuadro de texto y una etiqueta de texto estático que sólo le indica algo al usuario, el cuadro de texto muy probablemente deba ser accedido desde el código fuente para extraer el texto ingresado, mientras que para el texto de la etiqueta esto nunca haga falta. En ese caso, es el cuadro de texto el único componente que requiere id.

De todos los elementos que se pueden agregar a las escenas, un subconjunto de ellos están previstos especialmente para hacer interacción con el usuario, los cuales son colectivamente llamados controles. Encontramos bajo esta definición, por ejemplo, los botones, cuadros de texto, casillas de selección, etc., pero no se consideran controles los objetos contenedores, ya que el usuario no interactúa directamente con ellos.

En el archivo fxml se debe incluir un atributo del documento entero que indique el nombre completamente calificado de la clase controladora relacionada con la escena. En SceneBuilder ese atributo se establece en la solapa “Document”, sección “Controller”.

Para facilitar la creación de las clases controladoras existen dos mecanismos útiles que generan parte del código necesario. En el caso de NetBeans, haciendo click derecho en el archivo fxml se

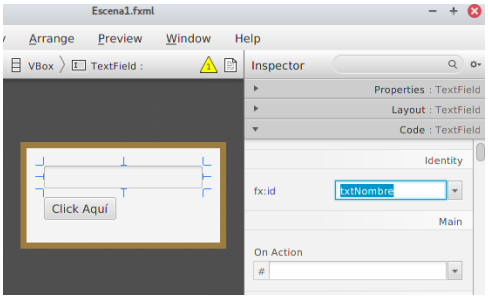

Figura 5: Asignación del fx:id

encuentra la opción “Make controller” que crea la clase controladora con el nombre indicado y agregando atributos para todos los controles que posean fx:id.

En el caso de utilizar otro IDE, SceneBuilder colabora mostrando el código de la controladora para poder copiarlo y pegarlo en el archivo correcto, que el programador debe crear por su cuenta. Para ello se selecciona en el menú “View” la opción “Show Sample Controller Skeleton”.

En el elemplo anterior, el único control que requiere fx:id es el cuadro de texto. Para establecerlo se lo debe seleccionar en el diseñador y asignarlo en la solapa “Inspector”, sección “Code”, como se ve en la figura 5. En este caso un nombre adecuado es `txtNombre`. El código generado por NetBeans tiene esta forma:

```
public class Escena1Controller implements Initializable {

    @FXML
    private TextField txtNombre;

    @Override
    public void initialize(URL location, ResourceBundle resources) {

    }

}
```

Manejadores de eventos

Los objetos pertenecientes a una escena pueden detectar la aparición de acciones externas que interactúen de alguna forma con ellos. Estas interacciones son denominadas colectivamente eventos. La mayoría de los eventos son generados por acción del usuario al realizar operaciones sobre los controles, tales como clicks o pulsaciones de teclas. Sin embargo el usuario no es el único agente externo a la aplicación que puede generar eventos; por ejemplo el sistema operativo, que suele generar eventos tales como la solicitud de apagado del programa a causa de un cierre de sesión o apagado o incluso otros programas en ejecución.

Los programas con interfaces gráficas deben responder ante la aparición de los eventos a medida que ocurren. A causa de esto la secuencia de ejecución de las diversas funcionalidades

que el programa ofrezca ya no van a se controlada por el programador (como en el caso de una aplicación de consola, donde el método `main()` inicia toda la secuencia), si no que el usuario elige en qué orden realiza cada acción. Para ello interactúa con los controles que la interfaz le muestra, seleccionando opciones de los menúes, presionando botones, escribiendo en cuadros de texto, etc.

Cada tipo de control puede atender un conjunto de eventos, de esta forma un botón puede atender el evento de click sobre sí mismo o una ventana puede detectar el intento de cierre. Sin embargo todos los tipos de controles poseen un evento que se considera más habitual o más probable, el cual es denominado "Action". La naturaleza del control determina cuál de todos los eventos es la acción principal, la cual normalmente es la más útil. En el caso de los botones, aunque se puedan identificar eventos tales como el ingreso del puntero del mouse a su superficie, o la pulsación de una tecla, el evento más probable es el de click. En el caso de un cuadro de texto, el evento más probable que sea necesario atender es el de la escritura de texto, y con esta lógica cada control define su acción principal.

Cuando se requiere que el programa atienda un evento, es decir que realice alguna tarea al ocurrir la acción esperada, el programador debe programar un método que va a ser invocado en forma automática, y casi nunca en forma determinista por el programa. Estos métodos son llamados **Manejadores de eventos** y reciben como único parámetro un objeto derivado de la clase `Event`. Este objeto posee información acerca del evento, incluyendo una referencia al control que lo detectó y cualquier otra información adicional relevante. Por ejemplo, en el caso de los eventos de click de mouse, posee atributos que indican con cual de los botones del mouse se hizo click o si se estaba pulsando simultáneamente una tecla modificadora (Shift, Alt, Ctrl).

Para agregar un controlador para un evento en particular se debe seleccionar el control en SceneBuilder y buscar el evento en la solapa "Inspector | Code". En el evento que se quiere atender únicamente debe escribirse el nombre del método manejador. A diferencia de otros lenguajes no existe ninguna restricción sobre el nombre del manejador, sin embargo es una práctica habitual que el nombre del manejador indique el nombre del control y la acción correspondiente. Así, para un botón llamado "btnAgregar", el manejador del evento "Action" podría denominarse `btnAgregarClick` o similar.

Luego de indicar el nombre del manejador se debe guardar la escena y volver a NetBeans para seleccionar "Make controller". Si el controlador ya existe NetBeans lo modifica sin eliminar el código existente. En este caso crea un nuevo método con el nombre seleccionado, un parámetro de tipo Event o de una de sus derivadas y la anotación @FXML.

Para agregar un manejador para el evento click del botón del ejemplo anterior se debe seleccionar desde el SceneBuilder el botón, y en la solapa "Code" indicar en "On Action" el nombre del método, por ejemplo `btnClick`. Luego de guardar la escena y regenerar el controlador se agrega un método `private void btnClick(ActionEvent event)`.

En el formulario del ejemplo, para que al presionar el botón se salude al usuario con el nombre que haya indicado el cuadro de texto, se puede obtener el mismo con el atributo text del control TextField. Luego para mostrar un cuadro de mensaje se utiliza la clase Alert que se verá en detalle más adelante. El código completo del controlador se presenta a continuación y la ejecución se muestra en la figura 6.

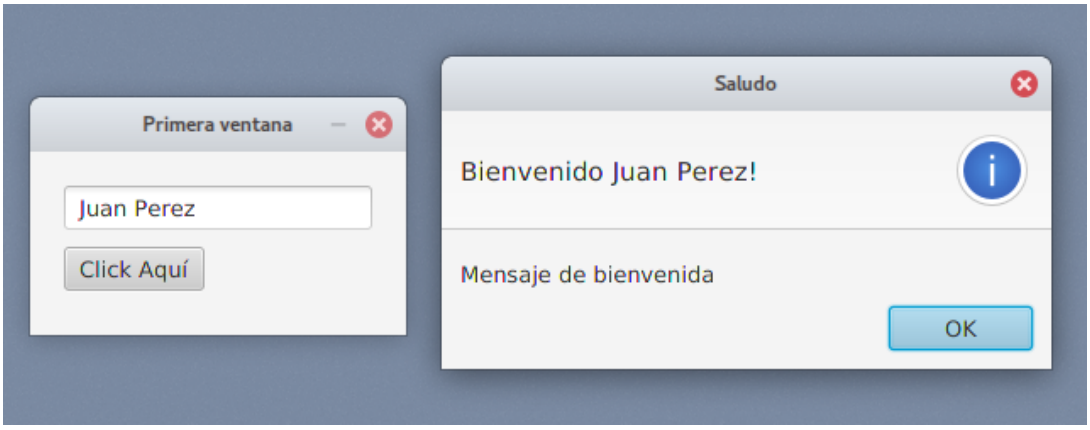

Figura 6: Ejecución con el evento click y un dialogo de alerta

```
public class Escena1Controller implements Initializable {

    @FXML
    private TextField txtNombre;

    @Override
    public void initialize(URL location, ResourceBundle resources) { }

    @FXML
    private void btnClick(ActionEvent event) {
        String nombre = txtNombre.getText();
        String saludo = "Bienvenido " + nombre + "!";

        Alert dialogo = new Alert(Alert.AlertType.INFORMATION);
        dialogo.setTitle("Saludo");
        dialogo.setHeaderText(saludo);
        dialogo.setContentText("Mensaje de bienvenida");
        dialogo.showAndWait();
    }
}
```