Escenarios

Con JavaFX las ventanas principales son instancias de la clase *Application*, la cual pertenece al paquete *javafx*. Para iniciar una aplicación se debe invocar al método *launch()* desde el método *main*. Al iniciarse, la aplicación llamará al método *start()*. Por lo general, el método *start()* crea y muestra una ventana de algún tipo.

También se deben definir los métodos *init()* y *stop()*. El método *init()* es automáticamente invocado por la aplicación para inicializar objetos antes que la aplicación inicie. Cuando la aplicación termina, la aplicación llamará inmediatamente al método *stop()* para cerrar archivos, liberar recursos, etc.

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

public class MyApplication extends Application {

    public void init() { ... }

    public void start(Stage escenario) { ... }

    public void stop() { ... }

    public static void main(String[] args) {
     launch(args);
    }
}
```

El método *start()* cuenta con un único parámetro de tipo *stage*, el cual es un objeto que representa un escenario (un tipo especial de ventana). Para hacer algo visible, se debe crear un objeto *Scene*, que define una escena (la apariencia de la aplicación). El objeto *Scene* contiene componentes gráficos denominados nodos y agrupados en contenedores que pertenecen a la clase *Pane*.

Un evento es algo que sucede en el programa en función de algún tipo de entrada que, generalmente, es causada por la interacción del usuario, como presionar una tecla en el teclado o hacer clic con el ratón. El origen de un evento es el componente para el que se generó el evento, es decir, cuando se gestionan clics de un botón, el componente botón es el origen. Un controlador de eventos es un procedimiento que contiene el código que se ejecutará cuando ocurra un tipo específico de evento en el programa.

Estos eventos se generan cuando el usuario interactúa con la GUI de la siguiente manera:

1. El usuario provoca un evento haciendo clic en un botón, presionando una tecla, seleccionando un elemento de la lista, etc.
2. Se generan eventos.
3. Se llama al controlador de eventos apropiado.
4. El código de manejo de eventos cambia el modelo (estados de los objetos) de alguna manera.
5. La interfaz de usuario se actualiza para reflejar estos cambios en el modelo.

Objetivos de aprendizaje

Al finalizar este ejercicio, tendrás la capacidad para:

- Crear una ventana gráfica utilizando JavaFX.
- Incorporar componentes gráficos a los contenedores de una ventana gráfica.

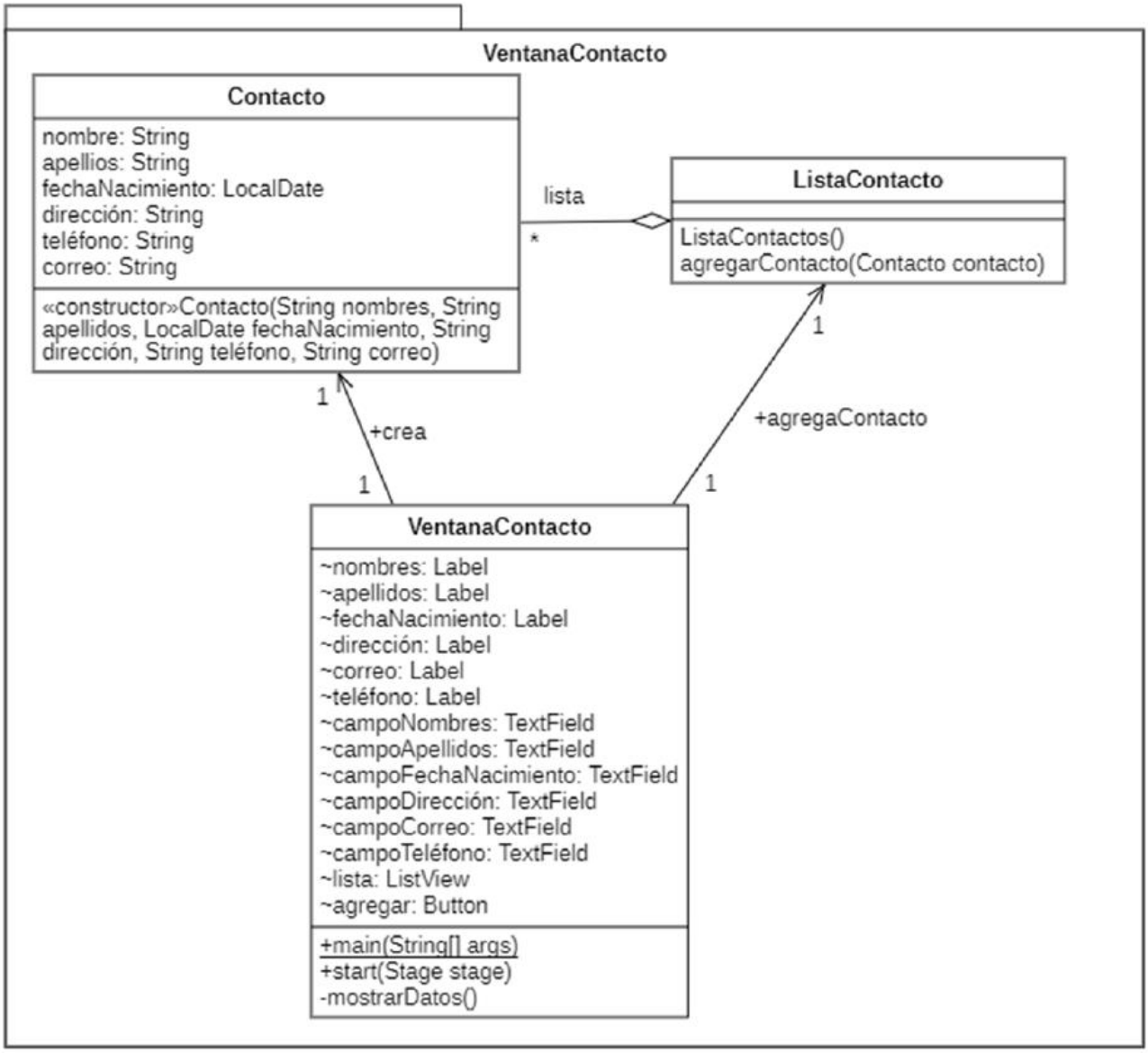
Enunciado: Contactos

Se requiere desarrollar un programa que permita (en una ventana) ingresar los datos de un contacto en una agenda personal. Los datos de un contacto son:

- Nombres y apellidos: se ingresan en objetos TextField independientes.
- Fecha de nacimiento: se debe desplegar un calendario utilizando la clase *DatePicker*.
- Dirección, teléfono, correo electrónico: se ingresa en un *TextField*.

Una vez se ingresan los datos, se debe oprimir un botón denominado “Agregar” que permite que el contacto se añada a una lista (clase *ListView*) ubicada en la parte inferior de la ventana.

Diagrama de clases



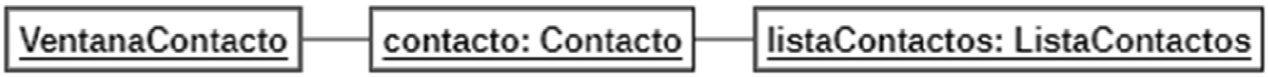
Explicación del diagrama de clases

El diagrama de clases presenta un solo paquete denominado “VentanaContacto”, que contiene tres clases: Contacto, ListaContacto y VentanaContacto. La clase Contacto tiene seis atributos: nombres y apellidos (de tipo *String*), fechaNacimiento (de tipo *LocalDate*), dirección, teléfono y correo (de tipo *String*). También cuenta con un constructor para inicializar estos atributos.

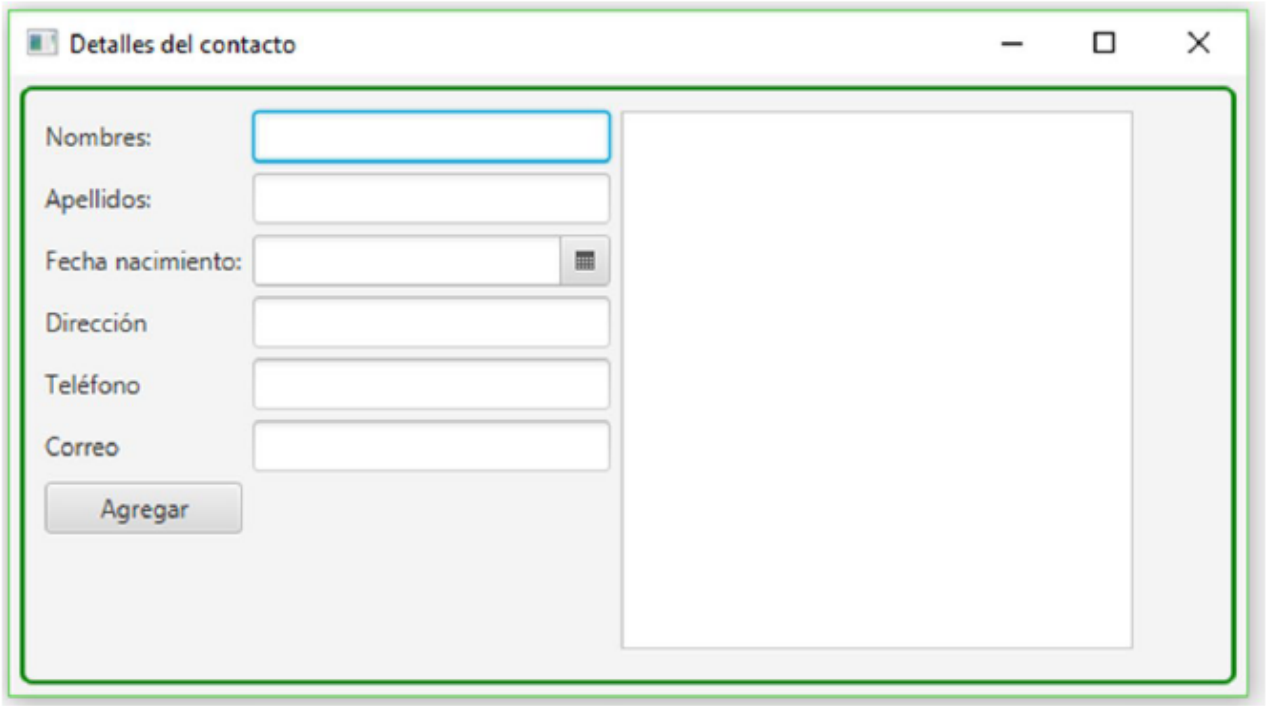
En primer lugar, la clase ListaContactos está relacionada con la clase Contacto por medio de una relación de agregación que se denota con una flecha con un rombo sin relleno en un extremo. Esta relación indica que una lista de contactos tiene asociada una colección de contactos con una multiplicidad de muchos (indicada con el asterisco *). La relación de agregación indica una relación con semántica débil entre las dos clases; de tal manera que, si la lista de contactos se elimina, los contactos siguen existiendo. En el código, esta relación de agregación se presenta como un atributo de la clase ListaContactos denominado lista, como indica el nombre de la relación de agregación.

En segundo lugar, la clase VentanaContacto tiene como atributos todos los componentes gráficos que conforman la ventana: etiquetas (*Label*), campos de ingreso de texto (*TextField*), un botón (*Button*) y una lista (*ListView*). También cuenta con los métodos: *start* que inicia la aplicación gráfica; el método *main* (punto de entrada a la aplicación); y el método mostrarDatos. El método *main* es un método estático, por lo cual se representa con su texto subrayado. La clase VentanaClase está relacionada con las dos clases anteriores ya que, durante su ejecución, se crea un contacto y dicho contacto se agrega a la lista de contactos. Los métodos *start* y *main* son públicos, lo cual se indica con el símbolo +. El método mostrarDatos es privado y se indica con el símbolo -.

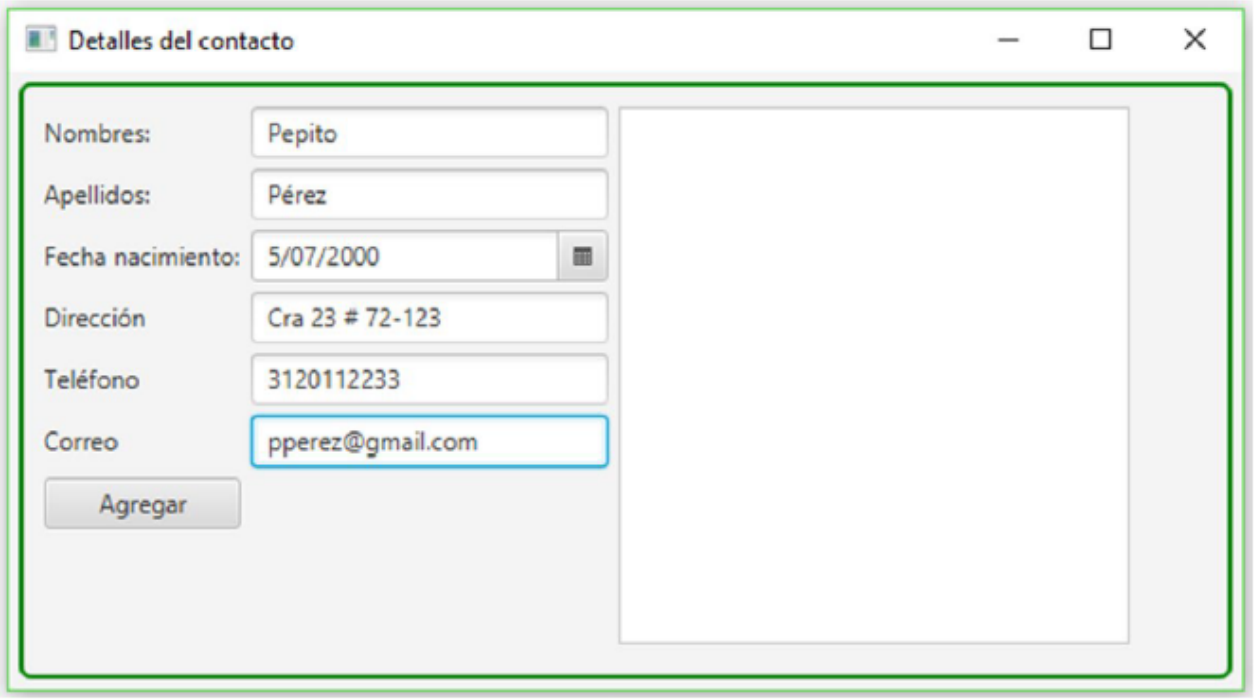
Diagrama de objetos



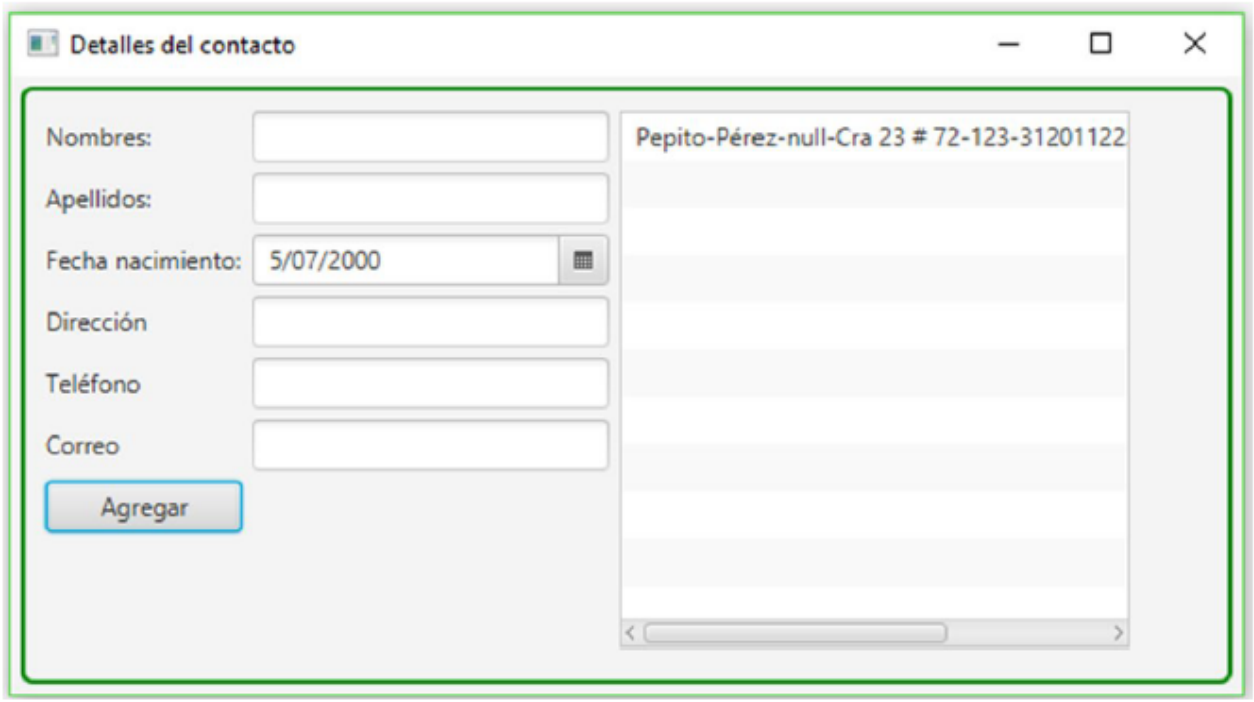
Ejecucion del programa



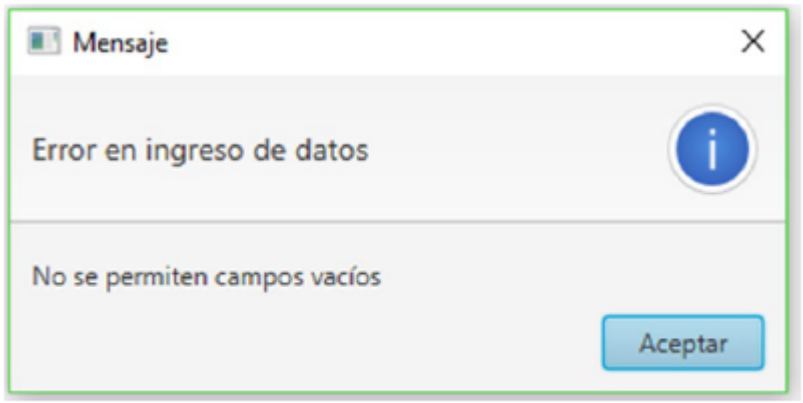
a) Ventana de contacto sin datos



b) Ingreso de datos



c) Contacto agregado a la lista



d) Mensaje de error

Ejercicios propuestos

- Realizar un programa que permita ingresar un valor numérico en una ventana y calcular: su logaritmo natural, su logaritmo en base 10, su raíz cuadrada y si es un número primo.
- Realizar un programa que permita convertir grados Fahrenheit a grados Celsius y viceversa utilizando JavaFX.
- Realizar el ejercicio 8.3. (figuras geométricas) utilizando JavaFX.

Instrucciones

Clase	Método	Descripción
Label	*Label()*	Constructor de la clase *Label*.
TextField	*TextField()*	Constructor de la clase *TextField*.
	String getText()	Obtiene el valor del texto ingresado.
	void setText(String valor)	Establece el valor del texto.
DatePicker	*DatePicker()*	Constructor de la clase *DatePicker*.
	LocalDate getValue()	Obtiene el valor seleccionado.
	void setValue(T valor)	Establece el valor del objeto.
ListView	*ListView()*	Constructor de la clase *ListView*.
	ObservableList<T> getItems()	Retorna una lista que contiene los elementos mostrados al usuario.
	void getItems().add(String valor)	Añade valores a la lista.
Button	*Button()*	Constructor de la clase *Button*.
	void setMaxWidth(Double.MAX_VALUE)	Establece la anchura del botón.
GridPane	*GridPane()*	Constructor de la clase *GridPane*.
	void setHgap(double valor)	Establece espacio horizontal entre componentes.
	void setVgap(double valor)	Establece espacio vertical entre componentes.
	void add(Component componente, fila, columna)	Añade un componente en la fila y columna especificada.
	void setStyle(String valor)	Establece un estilo CSS para el panel.
VBox	*VBox()*	Constructor de la clase VBox.
Stage	*Stage(Parent root, double anchura, double altura)*	Crea un objeto *Scene* para un nodo específico con un tamaño específico.
	void setScene(Scene valor)	Especifica la escena utilizada en este escenario.
	void setTitle()	Establece el título del escenario.
	void sizeToScene()	Establece el tamaño del escenario.
	void show()	Muestra la ventana.
Alert	*void setTitle(String valor)*	Establece título de la ventana del cuadro de diálogo.
	void setHeaderText(String valor)	Establece la cabecera del mensaje.
	void setContentText(String valor)	Establece el contenido de la ventana.
	void showAndWait()	Muestra el cuadro de diálogo.
Component	*setOnAction()*	Gestiona eventos de un componente.