UD09: Interfaz gráfica

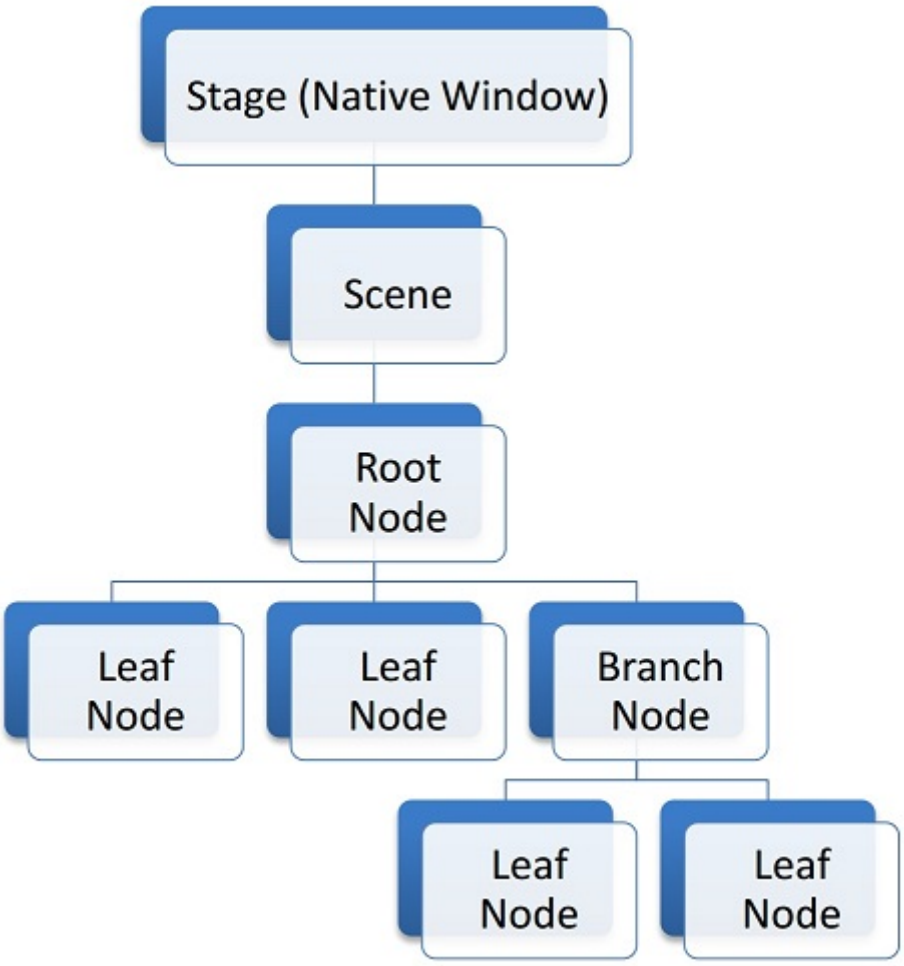
1. Introducción

El proyecto de documentación de JavaFX tiene como objetivo recopilar información útil para los desarrolladores de JavaFX de toda la web. El proyecto es de código abierto y fomenta la participación de la comunidad para garantizar que la documentación sea lo más pulida y útil posible.

2. Gráfico de escena

2.1. Descripción general

Un gráfico de escena es una estructura de datos de árbol que organiza (y agrupa) objetos gráficos para una representación lógica más sencilla. También permite que el motor de gráficos represente los objetos de la manera más eficiente al omitir total o parcialmente los objetos que no se verán en la imagen final. La siguiente figura muestra un ejemplo de la arquitectura del gráfico de escena JavaFX.

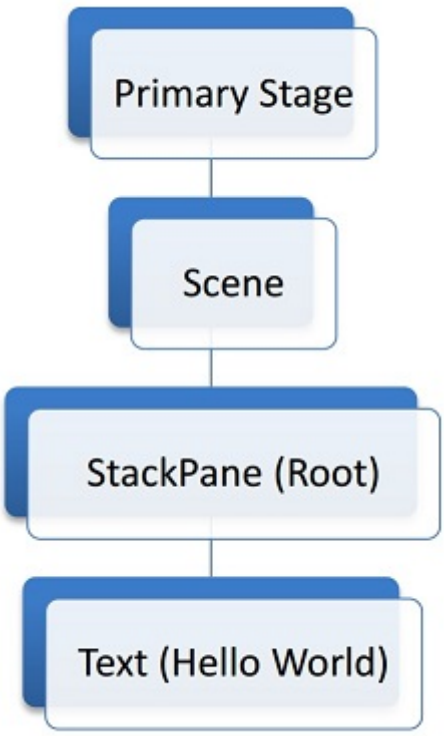


En la parte superior de la arquitectura hay un `Stage`. Una etapa es una representación JavaFX de una ventana de sistema operativo nativo. En un momento dado, un escenario puede tener un solo `Scene` adjunto. Una escena es un contenedor para el gráfico de escena JavaFX.

Todos los elementos en el gráfico de escena JavaFX se representan como `Node` objetos. Hay tres tipos de nudos: raíz, rama y hoja. El nodo raíz es el único nodo que no tiene un padre y está contenido directamente en una escena, que se puede ver en la figura anterior. La diferencia entre una rama y una hoja es que un nodo hoja no tiene hijos.

En el gráfico de escena, los nodos secundarios comparten muchas propiedades de un nodo principal. Por ejemplo, una transformación o un evento aplicado a un nodo padre también se aplicará recursivamente a sus hijos. Como tal, una jerarquía compleja de nodos se puede ver como un solo nodo para simplificar el modelo de programación. Exploraremos transformaciones y eventos en secciones posteriores.

En la siguiente figura se puede ver un ejemplo de un gráfico de escena "Hola Mundo".

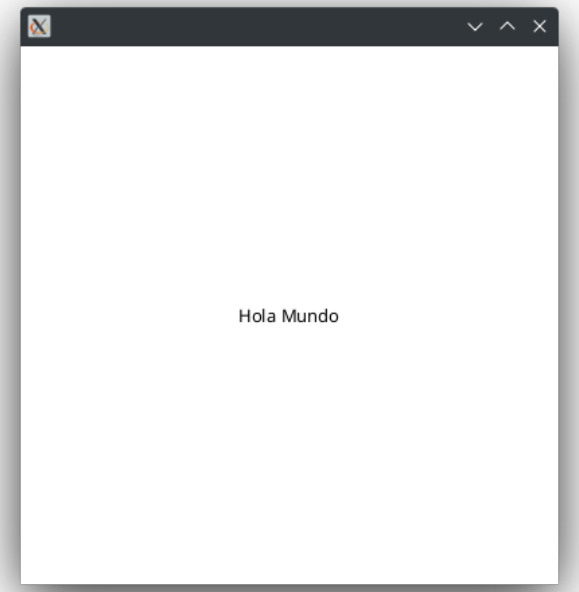


Una posible implementación que producirá un gráfico de escena que coincida con la figura anterior es la siguiente.

`E01_HolaMundo.java`

```java
import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class HolaMundo extends Application {

    private Parent createContent() {
        return new StackPane(new Text("Hola Mundo"));
    }

    @Override
    public void start(Stage stage) throws Exception {
        stage.setScene(new Scene(createContent(), 400, 400));
        stage.show();
    }

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

El resultado de ejecutar el código se ve en la siguiente figura.



Notas importantes:

- Un nodo puede tener un máximo de 1 padre.
- Un nodo en el gráfico de escena "activo" (adjunto a una escena actualmente visible) solo se puede modificar desde el subproceso de la aplicación JavaFX.

2.2. Transformaciones

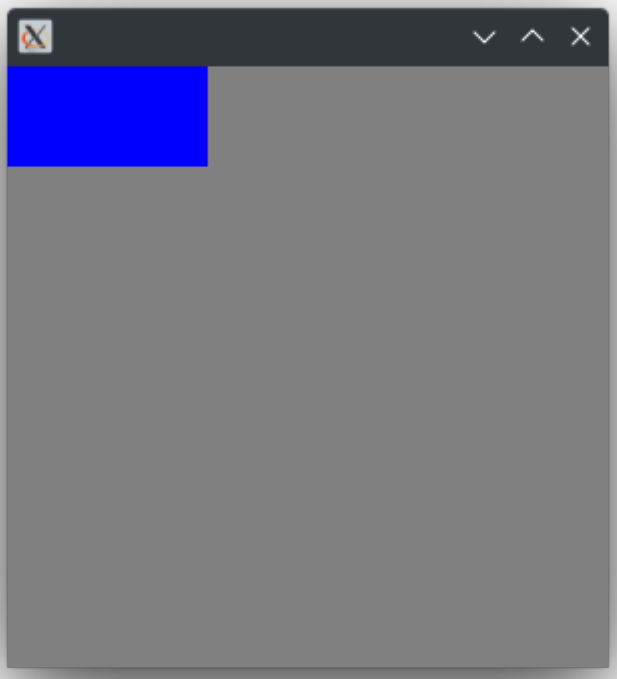
Usaremos la siguiente aplicación como ejemplo para demostrar las 3 transformaciones más comunes.

`E02_TransformApp.java`

```
1  import javafx.application.Application;
2  import javafx.scene.Parent;
3  import javafx.scene.Scene;
4  import javafx.scene.layout.Pane;
5  import javafx.scene.paint.Color;
6  import javafx.scene.shape.Rectangle;
7  import javafx.stage.Stage;
8
9  public class E02_TransformApp extends Application {
10
11     private Parent createContent() {
12         Rectangle box = new Rectangle(100, 50, Color.BLUE);
13         transform(box);
14         return new Pane(box);
15     }
16
```

```
17      private void transform(Rectangle box) {
18          // we will apply transformations here:
19
20          //Uncomment for translate
21          box.setTranslateX(100);
22          box.setTranslateY(200);
23
24          //uncomment for scale
25          box.setScaleX(1.5);
26          box.setScaleY(1.5);
27
28          //uncomment for rotate
29          box.setRotate(30);
30      }
31
32      @Override
33      public void start(Stage stage) throws Exception {
34          stage.setScene(new Scene(createContent(), 300, 300, Color.GRAY));
35          stage.show();
36      }
37
38      public static void main(String[] args) {
39          launch(args);
40      }
41  }
```

Ejecutar la aplicación dará como resultado la siguiente imagen.



En JavaFX, puede ocurrir una transformación simple en uno de los 3 ejes: X, Y o Z. La aplicación de ejemplo está en 2D, por lo que solo consideraremos los ejes X e Y.

2.2.1. Translación

En JavaFX y gráficos por computadora, `translate` significa moverse. Podemos trasladar nuestra caja en 100 píxeles en el eje X y 200 píxeles en el eje Y.

```
1  private void transform(Rectangle box) {
2      box.setTranslateX(100);
3      box.setTranslateY(200);
4  }
```

2.2.2. Escala

Puede aplicar la escala para hacer un nodo más grande o más pequeño. El valor de escala es una relación. Por defecto, un nodo tiene un valor de escala de 1 (100%) en cada eje. Podemos agrandar nuestra caja aplicando una escala de 1.5 en los ejes X e Y.

```
1  private void transform(Rectangle box) {
2      box.setScaleX(1.5);
3      box.setScaleY(1.5);
4  }
```

2.2.3. Rotación

La rotación de un nodo determina el ángulo en el que se representa el nodo. En 2D el único eje de rotación sensible es el eje Z. Giremos la caja 30 grados.

```
1  private void transform(Rectangle box) {
2      box.setRotate(30);
3  }
```

2.3. Manejo de eventos

Un evento notifica que ha ocurrido algo importante. Los eventos suelen ser lo "primitivo" de un sistema de eventos (también conocido como bus de eventos). Generalmente, un sistema de eventos tiene las siguientes 3 responsabilidades:

- `fire` (desencadenar) un evento,
- notificar `listeners` (a las partes interesadas) sobre el evento y
- `handle` (procesar) el evento.

El mecanismo de notificación de eventos lo realiza la plataforma JavaFX automáticamente. Por lo tanto, solo consideraremos cómo disparar eventos, escuchar eventos y cómo manejarlos.

Primero, vamos a crear un evento personalizado.

`E03_EventoUsuario.java`

```java
import javafx.event.Event;
import javafx.event.EventType;

public class E03_EventoUsuario extends Event {

    public static final EventType<E03_EventoUsuario> ANY = new EventType<>(Event.ANY, "ANY");

    public static final EventType<E03_EventoUsuario> LOGIN_SUCCEEDED = new EventType<>(ANY, "LOGIN_SUCCEEDED");

    public static final EventType<E03_EventoUsuario> LOGIN_FAILED = new EventType<>(ANY, "LOGIN_FAILED");

    public E03_EventoUsuario(EventType<? extends Event> eventType) {
        super(eventType);
    }
    // cualquier otro atributo importante como la fecha, la hora...
}
```

Dado que los tipos de eventos son fijos, generalmente se crean dentro del mismo archivo de origen que el evento. Podemos ver que hay 2 tipos específicos de eventos: `LOGIN_SUCCEEDED` y `LOGIN_FAILED`. Podemos escuchar estos tipos específicos de eventos:

```java
Node node = ...
node.addEventHandler(UserEvent.LOGIN_SUCCEEDED, event -> {
    // handle event
});
```

Alternativamente, podemos manejar cualquier `UserEvent`:

```java
Node node = ...
node.addEventHandler(UserEvent.ANY, event -> {
    // handle event
});
```

Finalmente, podemos construir y disparar nuestros propios eventos:

```java
UserEvent event = new UserEvent(UserEvent.LOGIN_SUCCEEDED);
Node node = ...
node.fireEvent(event);
```

Por ejemplo, `LOGIN_SUCCEEDED` o `LOGIN_FAILED` podría activarse cuando un usuario intenta iniciar sesión en una aplicación. Según el resultado del inicio de sesión, podemos permitir que el usuario acceda a la aplicación o bloquearlo. Si bien se puede lograr la misma funcionalidad con una `if` declaración simple, hay una ventaja significativa de un sistema de eventos. Los sistemas de eventos se diseñaron para permitir la comunicación entre varios módulos (subsistemas) en una aplicación sin acoplarlos estrechamente. Como tal, un sistema de audio puede reproducir un sonido cuando el

usuario inicia sesión. Por lo tanto, mantiene todo el código relacionado con el audio en su propio módulo. Sin embargo, no profundizaremos en los estilos arquitectónicos.

2.3.1. Eventos de entrada

Los eventos de teclado y ratón son los tipos de eventos más comunes utilizados en JavaFX. Cada `Node` proporciona los llamados "métodos de conveniencia" para manejar estos eventos. Por ejemplo, podemos ejecutar algún código cuando se presiona un botón:

```
Button button = ...
button.setOnAction(event -> {
    // button was pressed
});
```

Para mayor flexibilidad también podemos usar lo siguiente:

```
Button button = ...
button.setOnMouseEntered(e -> ...);
button.setOnMouseExited(e -> ...);
button.setOnMousePressed(e -> ...);
button.setOnMouseReleased(e -> ...);
```

El objeto `e` anterior es de tipo `MouseEvent` y se puede consultar para obtener información diversa sobre el evento, por ejemplo, `x` posiciones `y`, número de clics, etc. Finalmente, podemos hacer lo mismo con las teclas:

```
Button button = ...
button.setOnKeyPressed(e -> ...);
button.setOnKeyReleased(e -> ...);
```

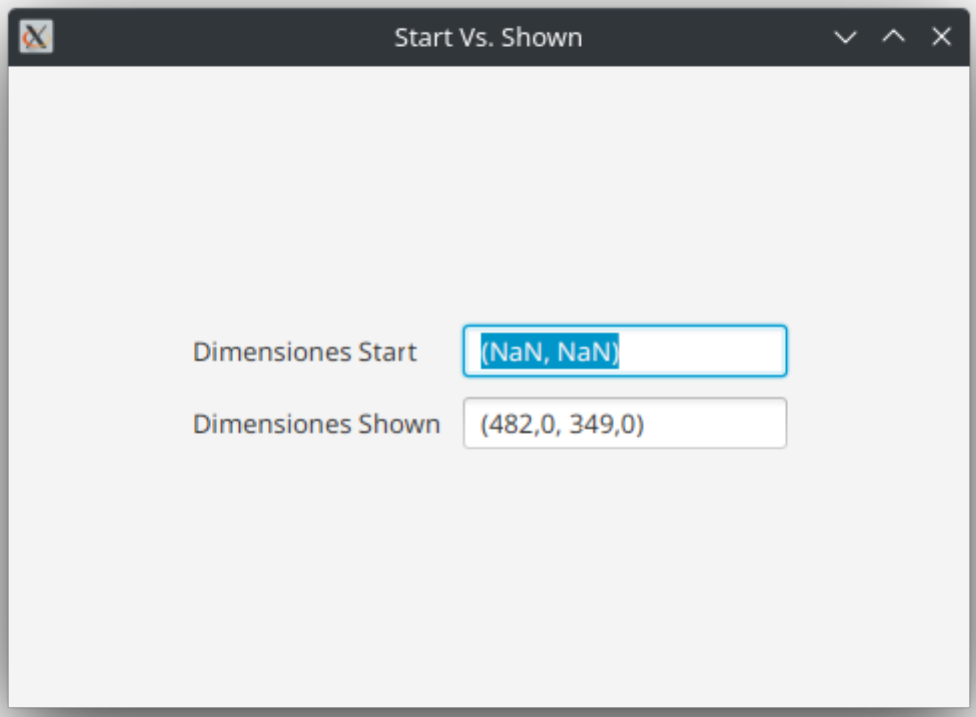
El objeto `e` aquí es de tipo `KeyEvent` y lleva información sobre el código de la tecla, que luego se puede asignar a una tecla física real en el teclado.

2.4. Sincronización

Es importante comprender la diferencia de tiempo entre la creación de controles de interfaz de usuario de JavaFX y la visualización de los controles. Al crear los controles de la interfaz de usuario, ya sea a través de la creación directa de objetos API o mediante FXML, es posible que te falten ciertos valores de geometría de pantalla, como las dimensiones de una ventana. Eso está disponible más tarde, en el instante en que se muestra la pantalla al usuario. Ese evento de visualización, llamado OnShown, es el momento en que se ha asignado una ventana y se completan los cálculos de diseño final.

Para demostrar esto, considere el siguiente programa que muestra las dimensiones de la pantalla mientras se crean los controles de la interfaz de usuario y las dimensiones de la pantalla cuando se muestra la pantalla. La siguiente captura de pantalla muestra la ejecución del programa. Cuando se crean los controles de la interfaz de usuario (`new VBox()`, `new Scene()`,

`primaryStage.setScene()`), no hay valores reales de alto y ancho de ventana disponibles como lo demuestran los valores "NaN" indefinidos.



Sin embargo, los valores de ancho y alto están disponibles una vez que se muestra la ventana. El programa registra un controlador de eventos para el evento `OnShown` y prepara la misma salida.

La siguiente es la clase Java del programa de demostración.

`E04_StartVsShown.java`

```
1  public class StartVsShownJavaFXApp extends Application {
2
3      private DoubleProperty startX = new SimpleDoubleProperty();
4      private DoubleProperty startY = new SimpleDoubleProperty();
5      private DoubleProperty shownX = new SimpleDoubleProperty();
6      private DoubleProperty shownY = new SimpleDoubleProperty();
7
8      @Override
9      public void start(Stage primaryStage) throws Exception {
10
11         Label startLabel = new Label("Start Dimensions");
12         TextField startTF = new TextField();
13         startTF.textProperty().bind(
14                 Bindings.format("(%.1f, %.1f)", startX, startY)
15         );
16
```

```
        Label shownLabel = new Label("Shown Dimensions");
        TextField shownTF = new TextField();
        shownTF.textProperty().bind(
                Bindings.format("(%.1f, %.1f)", shownX, shownY)
        );

        GridPane gp = new GridPane();
        gp.add( startLabel, 0, 0 );
        gp.add( startTF, 1, 0 );
        gp.add( shownLabel, 0, 1 );
        gp.add( shownTF, 1, 1 );
        gp.setHgap(10);
        gp.setVgap(10);

        HBox hbox = new HBox(gp);
        hbox.setAlignment(CENTER);

        VBox vbox = new VBox(hbox);
        vbox.setAlignment(CENTER);

        Scene scene = new Scene( vbox, 480, 320 );

        primaryStage.setScene( scene );

        // before show()...I just set this to 480x320, right?
        startX.set( primaryStage.getWidth() );
        startY.set( primaryStage.getHeight() );

        primaryStage.setOnShown( (evt) -> {
            shownX.set( primaryStage.getWidth() );
            shownY.set( primaryStage.getHeight() );  // all available now
        });

        primaryStage.setTitle("Start Vs. Shown");
        primaryStage.show();
    }

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

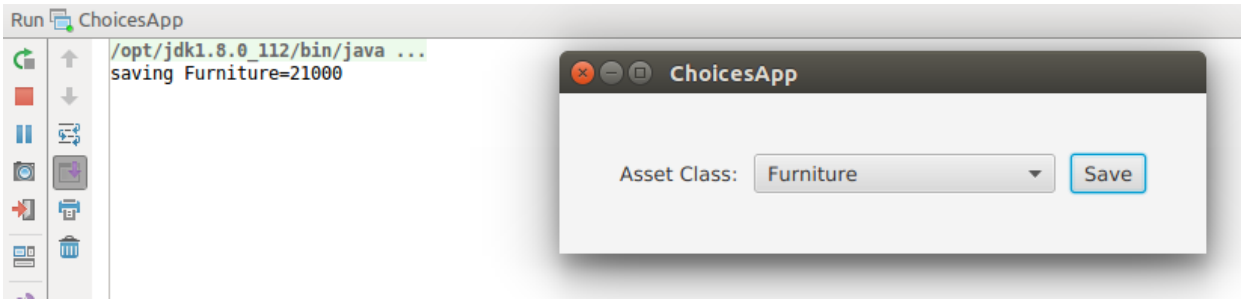
A veces, conocerá las dimensiones de la pantalla de antemano y puede usar esos valores en cualquier punto del programa JavaFX. Esto incluye antes del evento `OnShown`. Sin embargo, si su secuencia de inicialización contiene lógica que necesita estos valores, deberá trabajar con el evento `OnShown`. Un caso de uso podría ser trabajar con las últimas dimensiones guardadas o dimensiones basadas en la entrada del programa.

3. Controles de la interfaz de usuario

3.1. ChoiceBox

El `ChoiceBox` control es una lista de valores a partir de los cuales el usuario realiza una selección. En esta implementación particular, hay un valor vacío que hace que la selección sea opcional.

La siguiente captura de pantalla muestra la `ChoiceBox` aplicación. Se hace una selección de "Muebles" y `Button` se presiona Guardar. Si presionamos el `Button` `Guardar` invoca un `println()` que imprime el objeto.



El uso más simple del `ChoiceBox` es llenarlo con Strings. Este `ChoiceBox` se basa en una clase JavaFX llamada `Pair`. `Pair` es un contenedor general para cualquier par clave/valor y se puede usar en lugar de un dominio u otro objeto de propósito especial. Las cadenas solo deben usarse si pueden usarse sin manipulación o decodificarse de manera consistente.

3.1.1. Convertidor de cadenas

Cuando se usa un objeto complejo para respaldar un `ChoiceBox`, `StringConverter` se necesita a. Este objeto serializa un String hacia y desde el `ChoiceBox`. Para este programa, solo se necesita codificar toString(), que reemplaza el toString() predeterminado del `Pair` objeto. (Tanto toString como fromString necesitarán una implementación para compilar).

Se utiliza un objeto vacío EMPTY_PAIR para evitar NullPointerExceptions. Se puede acceder al valor devuelto de assetClass().getValue() y compararlo de manera consistente sin agregar una lógica especial de manejo de valores nulos.

El ChoiceBox se utiliza para seleccionar de una lista de valores. Cuando la lista de valores es de tipo complejo, proporcione un StringFormatter para serializar un objeto de lista en algo presentable. Si es posible, use un objeto vacío (en lugar de un nulo) para admitir valores opcionales.

3.1.2. Código completo

El código se puede probar en un solo archivo .java.

`ChoicesApp.java`

```
1  public class ChoicesApp extends Application {
2
3      private final ChoiceBox<Pair<String,String>> assetClass = new ChoiceBox<>();
4
5      private final static Pair<String, String> EMPTY_PAIR = new Pair<>("", "");
```

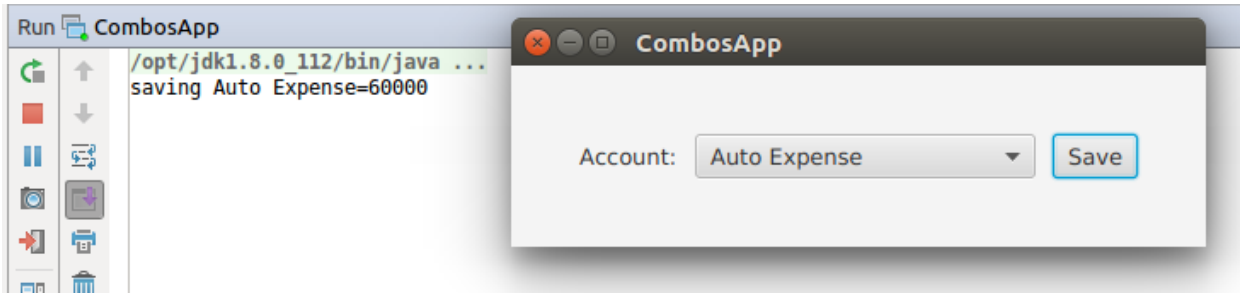
```
6
7       @Override
8       public void start(Stage primaryStage) throws Exception {
9
10          Label label = new Label("Asset Class:");
11          assetClass.setPrefWidth(200);
12          Button saveButton = new Button("Save");
13
14          HBox hbox = new HBox(
15                  label,
16                  assetClass,
17                  saveButton);
18          hbox.setSpacing( 10.0d );
19          hbox.setAlignment(Pos.CENTER );
20          hbox.setPadding( new Insets(40) );
21
22          Scene scene = new Scene(hbox);
23
24          initChoice();
25
26          saveButton.setOnAction(
27                  (evt) -> System.out.println("saving " + assetClass.getValue())
28          );
29
30          primaryStage.setTitle("ChoicesApp");
31          primaryStage.setScene( scene );
32          primaryStage.show();
33
34      }
35
36      private void initChoice() {
37
38          List<Pair<String,String>> assetClasses = new ArrayList<>();
39          assetClasses.add( new Pair("Equipment", "20000"));
40          assetClasses.add( new Pair("Furniture", "21000"));
41          assetClasses.add( new Pair("Investment", "22000"));
42
43          assetClass.setConverter( new StringConverter<Pair<String,String>>() {
44              @Override
45              public String toString(Pair<String, String> pair) {
46                  return pair.getKey();
47              }
48
49              @Override
50              public Pair<String, String> fromString(String string) {
51                  return null;
52              }
53          });
54
55          assetClass.getItems().add( EMPTY_PAIR );
56          assetClass.getItems().addAll( assetClasses );
57          assetClass.setValue( EMPTY_PAIR );
```

```
58
59      }
60
61      public static void main(String[] args) {
62          launch(args);
63      }
64  }
```

3.2. ComboBox

Un `ComboBox` es un control híbrido que presenta una lista de valores más un control de edición. Este artículo muestra una forma básica de la `ComboBox` cual es una lista no editable de elementos creados en una estructura de datos compleja.

Esta captura de pantalla muestra una ComboBoxApp que contiene una lista de cuentas de gastos. Las cuentas se almacenan en una clase JavaFX de clave/valor `Pair`. La consola muestra el resultado de una operación de guardado después de que el usuario seleccione "Gastos automáticos".



Este código agrega una etiqueta, un cuadro combinado y un botón a un HBox. El ComboBox se instancia como un campo y se inicializa en un método presentado más adelante initCombo(). Se coloca un controlador en el botón Guardar que genera un valor si se selecciona un elemento o un mensaje especial si no se selecciona ningún elemento.

3.2.1. CellFactory

El método initCombo() agrega varias cuentas de gastos a un archivo `List`. Esto `List` se agrega a los `ComboBox` elementos después de `Pair` agregar un objeto vacío. El valor inicial se establece en EMPTY_PAIR, que es una constante.

Si no se especifica, `ComboBox` utilizará el método toString() del objeto (en este artículo, a `Pair`) para representar un objeto de respaldo. Para cadenas, como una selección de "Sí" o "No", no se necesita código adicional. Sin embargo, toString() de a `Pair` generará tanto la clave legible por humanos como el valor preferido por la máquina. Los requisitos para esto `ComboBox` son usar solo las teclas legibles por humanos en la pantalla.

Para ello, se proporciona un cellFactory que configurará un `ListCell` objeto con la `Pair` clave como contenido. El `Callback` tipo es detallado, pero la esencia de la fábrica es establecer el texto de a `ListCell` en el método updateItem() de una clase interna anónima. Tenga en cuenta que se debe llamar al método de superclase.

se usa en el `Callback` método setButtonCell() para proporcionar una celda para el control de edición. Tenga en cuenta que este programa no es editable, que es el predeterminado. Sin embargo, se necesita factory.call(null), de lo contrario, solo se formateará correctamente el contenido del menú emergente y la vista del control en reposo recurrirá a toString().

Este artículo presentó un uso simple de `ComboBox`. Dado que este control no era editable, `ChoiceBox` se puede sustituir. Para representaciones gráficas no editables (por ejemplo, una forma codificada por colores para un valor de estado), `ComboBox` aún sería necesario definir el `Node` uso específico en el control.

3.2.2. Código completo

El código se puede probar en un solo archivo .java.

`CombosApp.java`

```
public class CombosApp extends Application {

    private final ComboBox<Pair<String, String>> account = new ComboBox<>();

    private final static Pair<String, String> EMPTY_PAIR = new Pair<>("", "");

    @Override
    public void start(Stage primaryStage) throws Exception {

        Label accountsLabel = new Label("Account:");
        account.setPrefWidth(200);
        Button saveButton = new Button("Save");

        HBox hbox = new HBox(
                accountsLabel,
                account,
                saveButton);
        hbox.setSpacing( 10.0d );
        hbox.setAlignment(Pos.CENTER );
        hbox.setPadding( new Insets(40) );

        Scene scene = new Scene(hbox);

        initCombo();

        saveButton.setOnAction( (evt) -> {
            if( account.getValue().equals(EMPTY_PAIR ) ) {
                System.out.println("no save needed; no item selected");
            } else {
                System.out.println("saving " + account.getValue());
            }
        });

        primaryStage.setTitle("CombosApp");
        primaryStage.setScene( scene );
```

```
36          primaryStage.show();
37      }
38
39      private void initCombo() {
40
41          List<Pair<String,String>> accounts = new ArrayList<>();
42
43          accounts.add( new Pair<>("Auto Expense", "60000") );
44          accounts.add( new Pair<>("Interest Expense", "61000") );
45          accounts.add( new Pair<>("Office Expense", "62000") );
46          accounts.add( new Pair<>("Salaries Expense", "63000") );
47
48          account.getItems().add( EMPTY_PAIR );
49          account.getItems().addAll( accounts );
50          account.setValue( EMPTY_PAIR );
51
52          Callback<ListView<Pair<String,String>>, ListCell<Pair<String,String>>> factory
=
53              (lv) ->
54                      new ListCell<Pair<String,String>>() {
55                          @Override
56                          protected void updateItem(Pair<String, String> item, boolean
empty) {
57                              super.updateItem(item, empty);
58                              if( empty ) {
59                                  setText("");
60                              } else {
61                                  setText( item.getKey() );
62                              }
63                          }
64                      };
65
66          account.setCellFactory( factory );
67          account.setButtonCell( factory.call( null ) );
68      }
69
70      public static void main(String[] args) {
71          launch(args);
72      }
73  }
```

3.3. ListView

3.3.1. Filtrado ListView en JavaFX

Este artículo demuestra cómo filtrar un ListView en una aplicación JavaFX. La aplicación gestiona dos listas. Una lista contiene todos los elementos del modelo de datos. La segunda lista contiene los elementos que se están visualizando actualmente. Un trozo de lógica de comparación almacenada como filtro media entre los dos.

El enlace se usa mucho para mantener las estructuras de datos sincronizadas con lo que el usuario ha seleccionado.

Esta captura de pantalla muestra la aplicación que contiene una fila superior de ToggleButtons que configuran el filtro y un ListView que contiene los objetos.

3.3.2. Estructuras de datos

El programa comienza con un modelo de dominio Player y una matriz de objetos Player.

La clase Player contiene un par de campos, team y playerName. Se proporciona un toString() para que cuando el objeto se agregue a ListView (que se presenta más adelante), no se necesite una clase ListCell personalizada.

Los datos de prueba para este ejemplo son una lista de jugadores de béisbol estadounidenses.

```
Player[] players = {new Player("BOS", "David Ortiz"),
                    new Player("BOS", "Jackie Bradley Jr."),
                    new Player("BOS", "Xander Bogarts"),
                    new Player("BOS", "Mookie Betts"),
                    new Player("HOU", "Jose Altuve"),
                    new Player("HOU", "Will Harris"),
                    new Player("WSH", "Max Scherzer"),
                    new Player("WSH", "Bryce Harper"),
                    new Player("WSH", "Daniel Murphy"),
                    new Player("WSH", "Wilson Ramos") };
```

3.3.3. Modelo

Como se mencionó al comienzo del artículo, el filtrado de ListView se centra en la gestión de dos listas. Todos los objetos se almacenan en una PlayerProperty de ObservableList envuelta y los objetos que se pueden ver actualmente se almacenan en una FilteredList envuelta, viewablePlayersProperty. viewablePlayersProperty se basa en playersProperty, por lo que las actualizaciones realizadas en los jugadores que cumplan con los criterios de FilteredList también se realizarán en viewablePlayers.

filterProperty() es una conveniencia para permitir que las personas que llaman se vinculen al Predicado subyacente.

La raíz de la interfaz de usuario es un VBox que contiene un HBox de ToggleButtons y un ListView.

3.3.4. Acción de filtrado

Se adjunta un controlador a los ToggleButtons que modificarán filterProperty. A cada ToggleButton se le proporciona un Predicado en el campo userData. toggleHandler utiliza este Predicado proporcionado al establecer la propiedad de filtro. Este código establece el caso especial "Mostrar todo" ToggleButton.

Los ToggleButtons que filtran un equipo específico se crean en tiempo de ejecución en función de la matriz Players. Este Stream hace lo siguiente.

1. Reduzca la lista de jugadores a una lista distinta de cadenas de equipo
2. Cree un ToggleButton para cada cadena de equipo
3. Establezca un Predicado para que cada ToggleButton se use como filtro
4. Recopile los ToggleButtons para agregarlos al contenedor HBox

3.3.5. Vista de la lista

El siguiente paso crea ListView y vincula ListView a viewablePlayersProperty. Esto permite que ListView reciba actualizaciones basadas en el cambio de filtro.

El resto del programa crea una Escena y muestra el Escenario. onShown carga el conjunto de datos en las listas playersProperty y viewablePlayersProperty. Aunque ambas listas están sincronizadas en esta versión particular del programa, si el filtro de stock es diferente a "sin filtro", no sería necesario modificar este código.

Este artículo usó el enlace para vincular una lista de objetos Player visibles a un ListView. Los jugadores visibles se actualizaron cuando se seleccionó un ToggleButton. La selección aplicó un filtro a un conjunto completo de jugadores que se mantuvo por separado como FilteredList (gracias @kleopatra_jx). El enlace se usó para mantener la interfaz de usuario sincronizada y para permitir una separación de preocupaciones en el diseño.

3.3.6. Código completo

```
public class FilterListApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        //
        // Test data
        //
        Player[] players = {new Player("BOS", "David Ortiz"),
                            new Player("BOS", "Jackie Bradley Jr."),
                            new Player("BOS", "Xander Bogarts"),
                            new Player("BOS", "Mookie Betts"),
                            new Player("HOU", "Jose Altuve"),
                            new Player("HOU", "Will Harris"),
                            new Player("WSH", "Max Scherzer"),
                            new Player("WSH", "Bryce Harper"),
                            new Player("WSH", "Daniel Murphy"),
                            new Player("WSH", "Wilson Ramos") };

        //
        // Set up the model which is two lists of Players and a filter criteria
        //
        ReadOnlyObjectProperty<ObservableList<Player>> playersProperty =
                new SimpleObjectProperty<>(FXCollections.observableArrayList());

        ReadOnlyObjectProperty<FilteredList<Player>> viewablePlayersProperty =
                new SimpleObjectProperty<FilteredList<Player>>(
                        new FilteredList<>(playersProperty.get()
                                ));

        ObjectProperty<Predicate<? super Player>> filterProperty =
            viewablePlayersProperty.get().predicateProperty();


        //
        // Build the UI
        //
        VBox vbox = new VBox();
        vbox.setPadding( new Insets(10));
        vbox.setSpacing(4);

        HBox hbox = new HBox();
        hbox.setSpacing( 2 );

        ToggleGroup filterTG = new ToggleGroup();

        //
        // The toggleHandler action wills set the filter based on the TB selected
        //
```

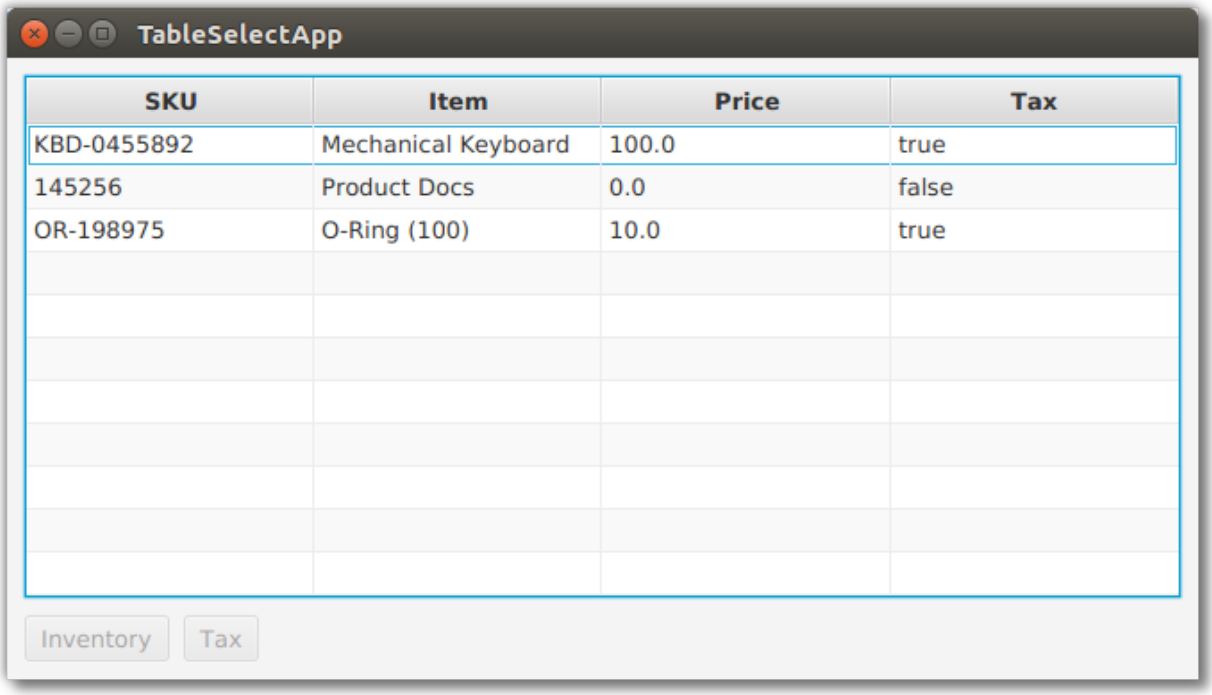
```
50          @SuppressWarnings("unchecked")
51          EventHandler<ActionEvent> toggleHandler = (event) -> {
52                  ToggleButton tb = (ToggleButton)event.getSource();
53                  Predicate<Player> filter = (Predicate<Player>)tb.getUserData();
54                  filterProperty.set( filter );
55              };
56
57          ToggleButton tbShowAll = new ToggleButton("Show All");
58          tbShowAll.setSelected(true);
59          tbShowAll.setToggleGroup( filterTG );
60          tbShowAll.setOnAction(toggleHandler);
61          tbShowAll.setUserData( (Predicate<Player>) (Player p) -> true);
62
63          //
64          // Create a distinct list of teams from the Player objects, then create
65          // ToggleButtons
66          //
67          List<ToggleButton> tbs = Arrays.asList( players)
68                  .stream()
69                  .map( (p) -> p.getTeam() )
70                  .distinct()
71                  .map( (team) -> {
72                      ToggleButton tb = new ToggleButton( team );
73                      tb.setToggleGroup( filterTG );
74                      tb.setOnAction( toggleHandler );
75                      tb.setUserData( (Predicate<Player>) (Player p) ->
team.equals(p.getTeam()) );
76                      return tb;
77                  })
78                  .collect(Collectors.toList());
79
80          hbox.getChildren().add( tbShowAll );
81          hbox.getChildren().addAll( tbs );
82
83          //
84          // Create a ListView bound to the viewablePlayers property
85          //
86          ListView<Player> lv = new ListView<>();
87          lv.itemsProperty().bind( viewablePlayersProperty );
88
89          vbox.getChildren().addAll( hbox, lv );
90
91          Scene scene = new Scene(vbox);
92
93          primaryStage.setScene( scene );
94          primaryStage.setOnShown((evt) -> {
95              playersProperty.get().addAll( players );
96          });
97
98          primaryStage.show();
99
100     }
```

```
101
102     public static void main(String args[]) {
103         launch(args);
104     }
105
106     static class Player {
107
108         private final String team;
109         private final String playerName;
110         public Player(String team, String playerName) {
111             this.team = team;
112             this.playerName = playerName;
113         }
114         public String getTeam() {
115             return team;
116         }
117         public String getPlayerName() {
118             return playerName;
119         }
120         @Override
121         public String toString() { return playerName + " (" + team + ")"; }
122     }
123 }
```

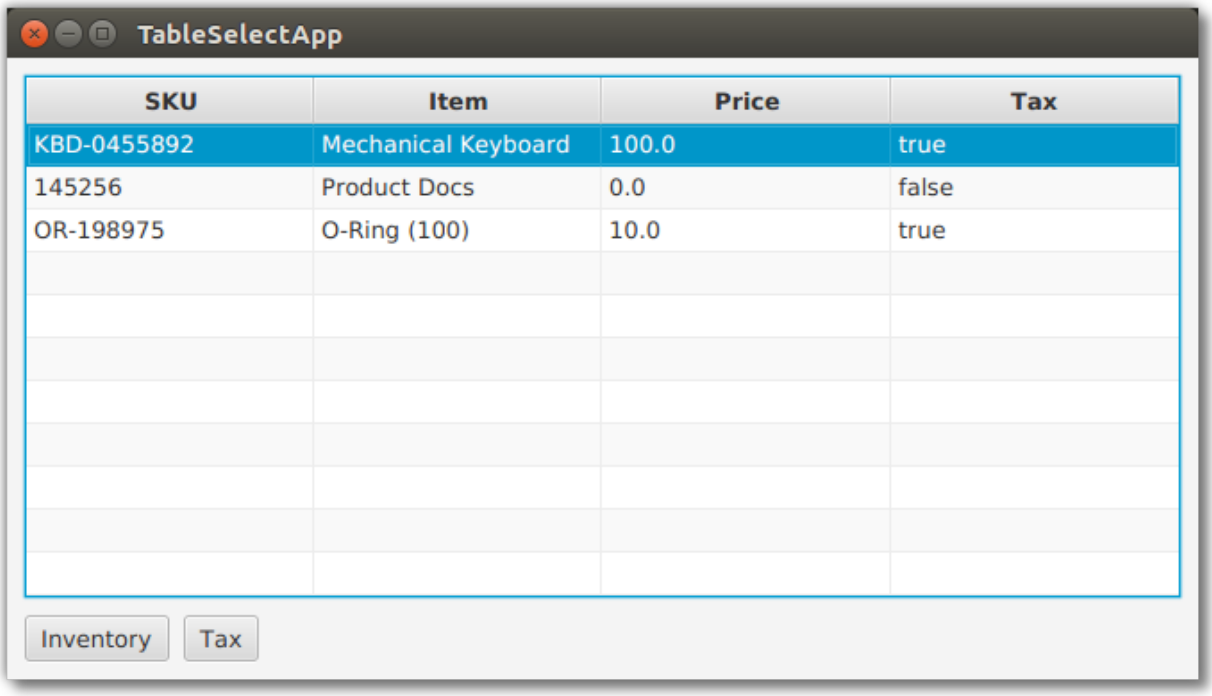
3.4. TableView

Para las aplicaciones comerciales de JavaFX, el `TableView` es un control esencial. Utilice a `TableView` cuando necesite presentar varios registros en una estructura plana de filas/columnas. Este ejemplo muestra los elementos básicos de a `TableView` y demuestra la potencia del componente cuando se aplica JavaFX Binding.

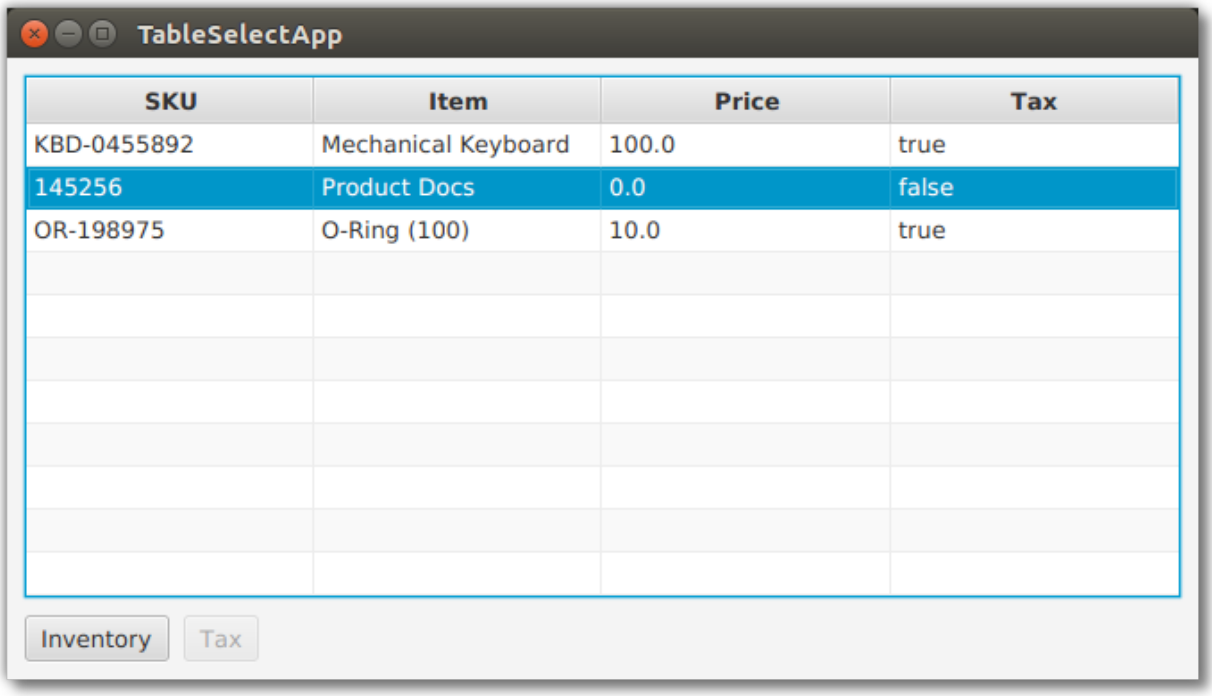
La aplicación de demostración es un `TableView`y un par de botones. Tiene cuatro TableColumns `TableView`: SKU, Artículo, Precio, Impuesto. Muestra `TableView`tres objetos en tres filas: teclado mecánico, documentos de productos, juntas tóricas. La siguiente captura de pantalla muestra la aplicación inmediatamente después del inicio.



La lógica deshabilitada de los botones se basa en las selecciones del archivo `TableView`. Inicialmente, no se seleccionan elementos, por lo que ambos botones están deshabilitados. Si se selecciona cualquier elemento, el primer elemento en la siguiente captura de pantalla, el Inventario `Button` está habilitado. `Button` También se habilita el Impuesto aunque requiere consultar el valor del Impuesto .



Si el valor del impuesto para el elemento seleccionado es falso, el impuesto `Button` se desactivará. Esta captura de pantalla muestra el segundo elemento seleccionado. El Inventario `Button` está habilitado pero el Impuesto `Button` no.



3.4.1. Modelo y Declaraciones

A `TableView` se basa en un modelo que es un POJO llamado Item.

Los `TableView` y `TableColumn` usan genéricos en sus declaraciones. Para `TableView`, el parámetro de tipo es Artículo. Para TableColumns, los parámetros de tipo son Item y el tipo de campo. El constructor de `TableColumn` acepta un nombre de columna. En este ejemplo, los nombres de las columnas difieren ligeramente de los nombres de los campos reales.

La adición de elementos de modelo a `TableView` se realiza mediante la adición de elementos a la colección subyacente.

En este punto, `TableView` se ha configurado y se han agregado los datos de prueba. Sin embargo, si fuera a ver el programa, vería tres filas vacías. Esto se debe a que a JavaFX le falta el vínculo entre POJO y TableColumns. Ese enlace se agrega a TableColumns mediante cellValueFactory.

En este punto mostrará los datos en las columnas correspondientes.

3.4.2. Selección

Para recuperar el elemento o elementos seleccionados en un `TableView`, use el objeto selectionModel separado. Llamar a tblItems.getSelectionModel() devuelve un objeto que incluye una propiedad "selectedItem". Esto se puede recuperar y usar en un método, por ejemplo, para abrir una pantalla de detalles de edición. Como alternativa, getSelectionModel() puede devolver una propiedad JavaFX "selectedItemProperty" para expresiones vinculantes.

En la aplicación de demostración, dos botones están vinculados al modelo de selección del archivo `TableView`. Sin enlace, puede agregar oyentes que examinen la selección y realicen una llamada como setDisabled() en un botón. Antes de la `TableView` selección, también necesitaría lógica de inicialización para manejar el caso en el que no hay selección. La sintaxis de vinculación expresa

esta lógica en una declaración declarativa que puede manejar tanto el oyente como la inicialización en una sola línea.

La propiedad de desactivación de btnInventory será verdadera si no hay ningún elemento seleccionado (isNull()). Cuando se muestra la pantalla por primera vez, no se realiza ninguna selección y `Button` se desactiva. Una vez que se realiza cualquier selección, se habilita btnInventory (deshabilitar=falso).

la lógica btnCalcTax es un poco más compleja. btnCalcTax también está deshabilitado cuando no hay selección. Sin embargo, btnCalcTax también considerará el contenido del elemento seleccionado. Se usa un enlace compuesto or() para unir estas dos condiciones. Como antes, hay una expresión isNull() para no seleccionar. Bindings.select() comprueba el valor de Item.taxable. Un artículo gravable verdadero habilitará btnCalcTax mientras que un artículo falso deshabilitará el `Button`.

Bindings.select() es el mecanismo para extraer un campo de un objeto. selectedItemProperty() es el elemento seleccionado cambiante y "sujeto a impuestos" es la ruta de acceso de un solo salto al campo sujeto a impuestos.

Este ejemplo mostró cómo configurar un `TableView` basado en un POJO. También presentaba un par de poderosas expresiones vinculantes que le permiten vincular controles relacionados sin escribir oyentes adicionales ni código de inicialización. Es `TableView` un control indispensable para el desarrollador de aplicaciones empresariales JavaFX. Será el mejor y más conocido control para mostrar una lista de elementos estructurados.

3.4.3. Código completo

`Item.java`

```
1  public class Item {
2
3      private final String sku;
4      private final String descr;
5      private final Float price;
6      private final Boolean taxable;
7
8      public Item(String sku, String descr, Float price, Boolean taxable) {
9          this.sku = sku;
10         this.descr = descr;
11         this.price = price;
12         this.taxable = taxable;
13     }
14
15     public String getSku() {
16         return sku;
17     }
18
19     public String getDescr() {
20         return descr;
21     }
22
23     public Float getPrice() {
```

```
        return price;
    }

    public Boolean getTaxable() {
        return taxable;
    }
}
```

`TableSelectApp.java`

```
public class TableSelectApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        TableView<Item> tblItems = new TableView<>();
        tblItems.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);

        VBox.setVgrow(tblItems, Priority.ALWAYS );

        TableColumn<Item, String> colSKU = new TableColumn<>("SKU");
        TableColumn<Item, String> colDescr = new TableColumn<>("Item");
        TableColumn<Item, Float> colPrice = new TableColumn<>("Price");
        TableColumn<Item, Boolean> colTaxable = new TableColumn<>("Tax");

        colSKU.setCellValueFactory( new PropertyValueFactory<>("sku") );
        colDescr.setCellValueFactory( new PropertyValueFactory<>("descr") );
        colPrice.setCellValueFactory( new PropertyValueFactory<>("price") );
        colTaxable.setCellValueFactory( new PropertyValueFactory<>("taxable") );

        tblItems.getColumns().addAll(
            colSKU, colDescr, colPrice, colTaxable
        );

        tblItems.getItems().addAll(
            new Item("KBD-0455892", "Mechanical Keyboard", 100.0f, true),
            new Item( "145256", "Product Docs", 0.0f, false ),
            new Item( "OR-198975", "O-Ring (100)", 10.0f, true)
        );

        Button btnInventory = new Button("Inventory");
        Button btnCalcTax = new Button("Tax");

        btnInventory.disableProperty().bind(
            tblItems.getSelectionModel().selectedItemProperty().isNull()
        );

        btnCalcTax.disableProperty().bind(
            tblItems.getSelectionModel().selectedItemProperty().isNull().or(
                    Bindings.select(
                        tblItems.getSelectionModel().selectedItemProperty(),
```

```
                        "taxable"
                    ).isEqualTo(false)
            )
        );

        HBox buttonHBox = new HBox( btnInventory, btnCalcTax );
        buttonHBox.setSpacing( 8 );

        VBox vbox = new VBox( tblItems, buttonHBox );
        vbox.setPadding( new Insets(10) );
        vbox.setSpacing( 10 );

        Scene scene = new Scene(vbox);

        primaryStage.setTitle("TableSelectApp");
        primaryStage.setScene( scene );
        primaryStage.setHeight( 376 );
        primaryStage.setWidth( 667 );
        primaryStage.show();
    }

    public static void main(String[] args) {

        launch(args);
    }
}
```

3.5. ImageView

JavaFX proporciona las clases `Image` y `ImageView` para mostrar imágenes gráficas BMP, GIF, JPEG y PNG. Imagen es una clase que contiene los bytes de la imagen y, opcionalmente, la información de escala. El objeto Imagen se carga mediante un subproceso en segundo plano y la clase Imagen proporciona métodos para interactuar con la operación de carga. El objeto Imagen se usa independientemente de ImageView para crear cursores e íconos de aplicaciones.

ImageView es un JavaFX `Node` que contiene un objeto de imagen. ImageView hace que una imagen esté disponible en todo el marco. Un ImageView se puede agregar a un contenedor solo o junto con otros controles de IU. Por ejemplo, se puede agregar una imagen `Label` configurando la propiedad gráfica de la etiqueta.

Las imágenes también se pueden mostrar y manipular usando JavaFX CSS.

Esta captura de pantalla muestra un `TilePane` que contiene cuatro mosaicos del mismo tamaño. Cada mosaico contiene un ImageView de un teclado.

La imagen superior izquierda se muestra con el tamaño de imagen original de 320x240. La imagen superior derecha está escalada proporcionalmente. Dado que la imagen superior derecha es un rectángulo y el mosaico que lo contiene es un cuadrado, hay espacios en la parte superior e inferior para mantener la proporción correcta al estirar el ancho.

La imagen inferior izquierda llena el contenedor por completo. Sin embargo, al hacer que la imagen rectangular se ajuste al contenedor cuadrado, la imagen no se escala proporcionalmente sino que se estira en ambas direcciones.

La imagen inferior derecha llena el contenedor utilizando una versión ampliada de la imagen. Se crea una ventana gráfica cuadrada a partir de un Rectangle2D de 100x100 y se amplía proporcionalmente. Si bien la imagen de baja calidad es borrosa, no se deforma.

3.5.1. Imagen

La clase de imagen proporciona constructores para crear un objeto de imagen a partir de las dimensiones del archivo de imagen o de un objeto transformado. Estas tres llamadas al constructor crean los objetos Image que se usan en los mosaicos de arriba a la derecha, de abajo a la izquierda y de abajo a la derecha, respectivamente.

ImageApp.java

```
public class ImageApp extends Application {

    private final static String IMAGE_LOC = "images/keyboard.jpg";

    @Override
    public void start(Stage primaryStage) throws Exception {

        Image image2 = new Image(IMAGE_LOC, 360.0d, 360.0d, true, true );
        Image image3 = new Image(IMAGE_LOC, 360.0d, 360.0d, false, true);
        Image image4 = new Image(IMAGE_LOC);
```

La URL de cadena que se pasa a todas las formas del constructor de imágenes es relativa a la ruta de clase. También se puede usar una URL absoluta como "https://www.bekwam.com/images/bekwam_logo_hdr_rounded.png". Tenga en cuenta que las URL absolutas no generarán un error si no se encuentra su recurso.

imagen2 e imagen3 especifican dimensiones, formando un cuadrado más grande que el rectángulo de la imagen original. image2 conservará la relación de aspecto ("verdadera"). El constructor de image3 no conserva la relación de aspecto y aparecerá estirado.

3.5.2. Vista de imagen

ImageView es un contenedor de nodo que permite que el objeto de imagen se use en contenedores JavaFX y controles de interfaz de usuario. En la imagen superior izquierda, se usa una forma abreviada de ImageView que pasa solo la URL de la imagen. Respetará las dimensiones originales y no requiere un objeto de imagen adicional.

iv3 e iv3 se basan en los objetos image2 e image3. Recuerde que estos objetos produjeron imágenes transformadas que se ajustan al contenedor cuadrado.

iv4 también se basa en un objeto de imagen transformado, pero en el caso de iv4, la transformación se realiza a través del objeto ImageView en lugar de la imagen. ImageView.setFitHeight se llama en lugar de Image.setFitHeight.

Además, se ajusta la ventana gráfica de iv4. Viewport controla la parte visible de ImageView. En este caso, la ventana gráfica se define como una sección de 100x100 de la imagen desplazada 20 píxeles a la izquierda y 50 píxeles hacia arriba.

Esta sección mostró las clases Image e ImageView que se usan para mostrar una imagen en un contenedor u otro control de UI. Estas clases definen el comportamiento de escalado de la imagen y se pueden usar con una ventana gráfica Rectangle2D para brindar una personalización adicional de la visualización de la imagen.

3.5.3. Fuente

El código fuente completo y el proyecto Gradle se pueden encontrar en el siguiente enlace.

Código postal de origen de ImageApp

4. Diseño

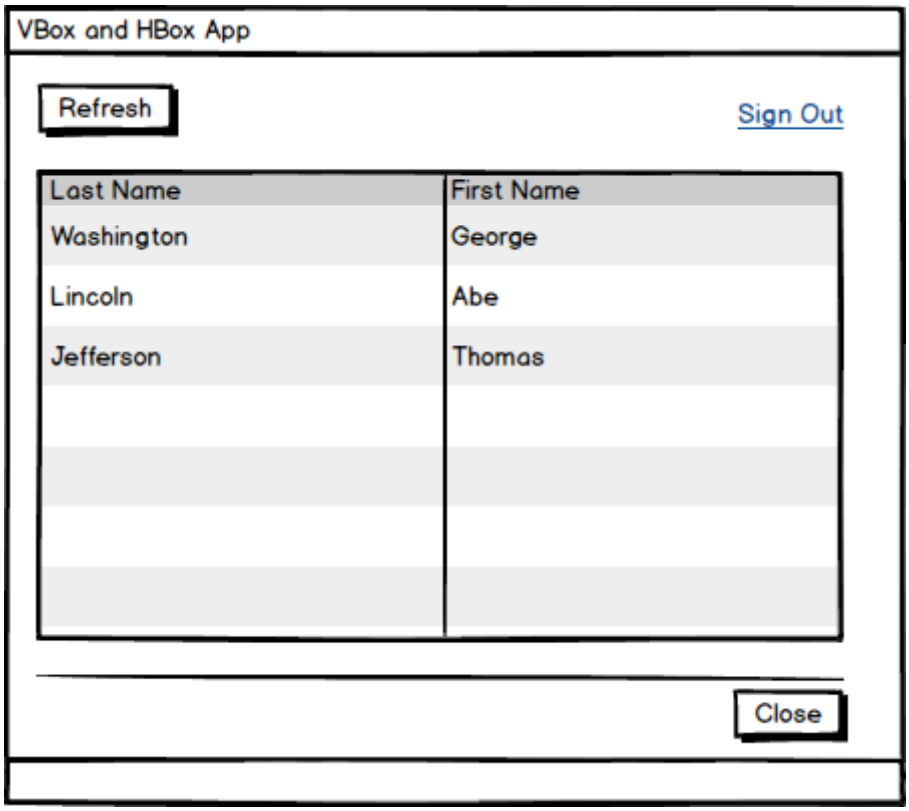
4.1. VBox y HBox

El diseño en JavaFX comienza con la selección de los controles de contenedor correctos. Los dos controles de diseño que uso con más frecuencia son `VBox` y `HBox`. `VBox` es un contenedor que organiza a sus hijos en una pila vertical. `HBox` ordena a sus hijos en una fila horizontal. El poder de estos dos controles proviene de envolverlos y establecer algunas propiedades clave: alineación, hgrow y vgrow.

Este artículo demostrará estos controles a través de un proyecto de ejemplo. Una maqueta del proyecto muestra una interfaz de usuario con lo siguiente:

- Una fila de controles superiores que contiene Actualizar `Button` y Cerrar sesión `Hyperlink`,
- A `TableView` que crecerá para ocupar el espacio vertical adicional, y
- Un cierre `Button`.

La interfaz de usuario también presenta un `Separator` panel que divide la parte superior de la pantalla con lo que puede convertirse en un panel inferior estándar (Guardar `Button`, Cancelar `Button`, etc.) para la aplicación.



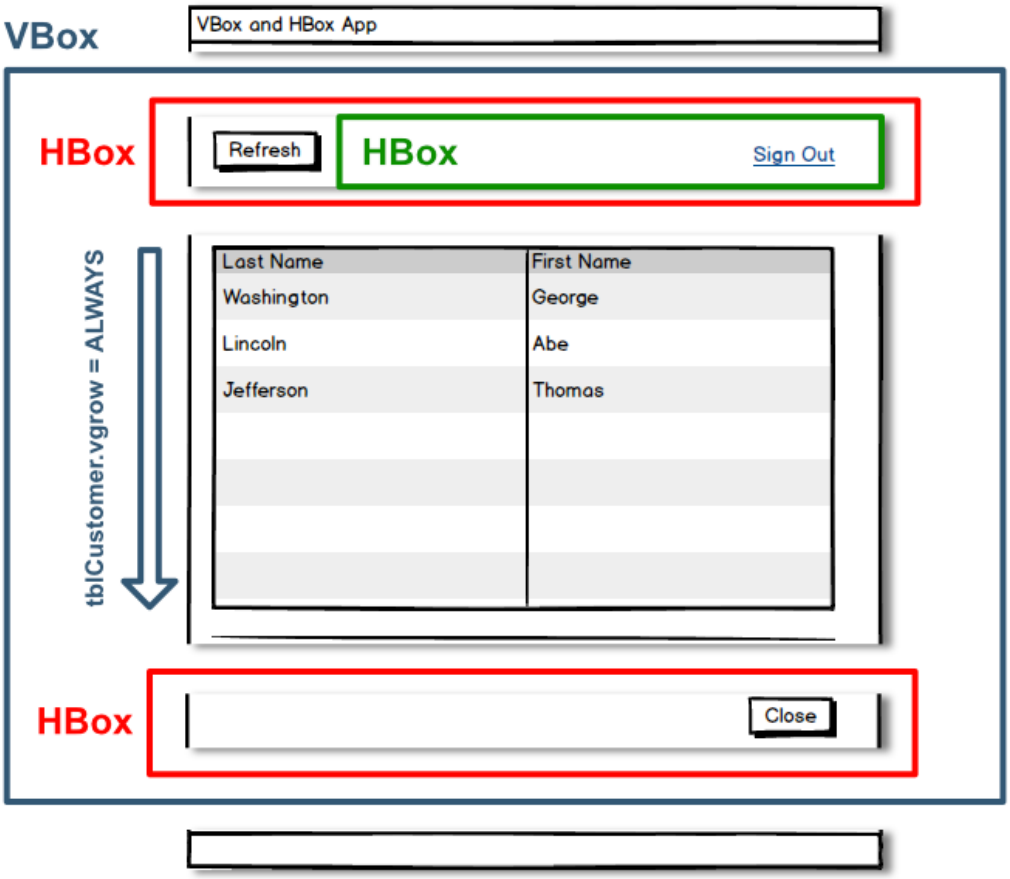
4.1.1. Estructura

A `VBox` es el contenedor más externo "vbox". Este será el `Parent` proporcionado a la Escena. El simple hecho de colocar los controles de la interfaz de usuario en esto `VBox` permitirá que los controles, sobre todo el `TableView`, se estiren para adaptarse al espacio horizontal disponible. Los controles superiores, Actualizar `Button` y Cerrar sesión `Hyperlink`, están envueltos en un archivo

`HBox`. Del mismo modo, envuelvo el cierre inferior `Button` en un `HBox`, lo que permite botones adicionales.

```
VBox vbox = new VBox();

Button btnRefresh = new Button("Refresh");

HBox topRightControls = new HBox();
topRightControls.getChildren().add( signOutLink );

topControls.getChildren().addAll( btnRefresh, topRightControls );

TableView<Customer> tblCustomers = new TableView<>();
Separator sep = new Separator();

HBox bottomControls = new HBox();

Button btnClose = new Button("Close");

bottomControls.getChildren().add( btnClose );

vbox.getChildren().addAll(
        topControls,
        tblCustomers,
        sep,
        bottomControls
);
```

Esta imagen muestra la maqueta desglosada por contenedor. El padre `VBox` es el rectángulo azul más externo. Los HBoxes son los rectángulos interiores (rojo y verde).



4.1.2. Alineación y Hgrow

Actualizar `Button` está alineado a la izquierda mientras que Cerrar sesión `Hyperlink` está alineado a la derecha. Esto se logra usando dos HBoxes. topControls es un `HBox` que contiene Actualizar `Button` y también contiene un `HBox` con Cerrar sesión `Hyperlink`. A medida que la pantalla se hace más ancha, Cerrar sesión `Hyperlink` se desplazará hacia la derecha, mientras que Actualizar `Button` mantendrá su alineación izquierda.

La alineación es la propiedad que le dice a un contenedor dónde colocar un control. topControls establece la alineación en BOTTOM_LEFT. topRightControls establece la alineación con BOTTOM_RIGHT. "BOTTOM" se asegura de que la línea de base del texto "Actualizar" coincida con la línea de base del texto "Cerrar sesión".

Para que el cierre de sesión `Hyperlink` se mueva hacia la derecha cuando la pantalla se ensancha, `Priority.ALWAYS` es necesario. Esta es una señal para que JavaFX amplíe topRightControls. De lo contrario, topControls mantendrá el espacio y topRightControls aparecerá a la izquierda. Cerrar sesión `Hyperlink` todavía estaría alineado a la derecha pero en un contenedor más estrecho.

Tenga en cuenta que `setHgrow()` es un método estático y no se invoca en topControls `HBox` ni en sí mismo, topRightControls. Esta es una faceta de la API de JavaFX que puede resultar confusa porque la mayoría de las API establece propiedades a través de setters en objetos.

```
topControls.setAlignment( Pos.BOTTOM_LEFT );

HBox.setHgrow(topRightControls, Priority.ALWAYS );
topRightControls.setAlignment( Pos.BOTTOM_RIGHT );
```

Close `Button` se envuelve en un `HBox` y se posiciona usando la prioridad BOTTOM_RIGHT.

```
bottomControls.setAlignment(Pos.BOTTOM_RIGHT );
```

4.1.3. crecer

Dado que el contenedor más externo es `VBox`, el niño `TableView` se expandirá para ocupar espacio horizontal adicional cuando se amplíe la ventana. Sin embargo, cambiar el tamaño vertical de la ventana producirá un espacio en la parte inferior de la pantalla. `VBox` no cambia automáticamente el tamaño de ninguno de sus elementos secundarios . Al igual que con topRightControls `HBox`, se puede configurar un indicador de crecimiento. En el caso del `HBox`, se trataba de una instrucción de cambio de tamaño horizontal setHgrow(). Para el `TableView` contenedor `VBox`, será setVgrow().

```
VBox.setVgrow( tblCustomers, Priority.ALWAYS );
```

4.1.4. Margen

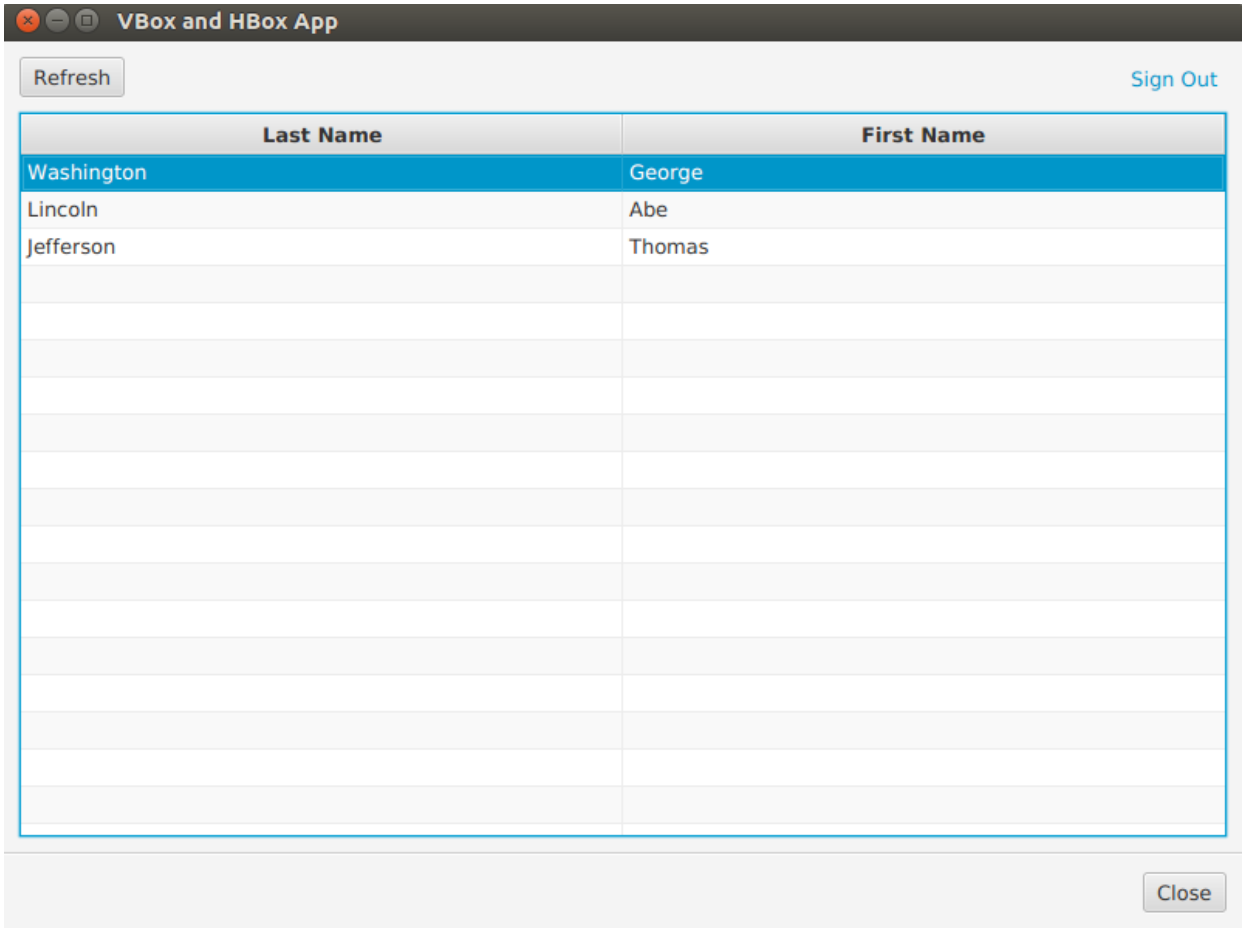
Hay algunas formas de espaciar los controles de la interfaz de usuario. Este artículo usa la propiedad margin en varios de los contenedores para agregar espacios en blanco alrededor de los controles. Estos se configuran individualmente en lugar de usar un espacio en el `VBox` para que el Separador abarque todo el ancho.

```
VBox.setMargin( topControls, new Insets(10.0d) );
VBox.setMargin( tblCustomers, new Insets(0.0d, 10.0d, 10.0d, 10.0d) );
VBox.setMargin( bottomControls, new Insets(10.0d) );
```

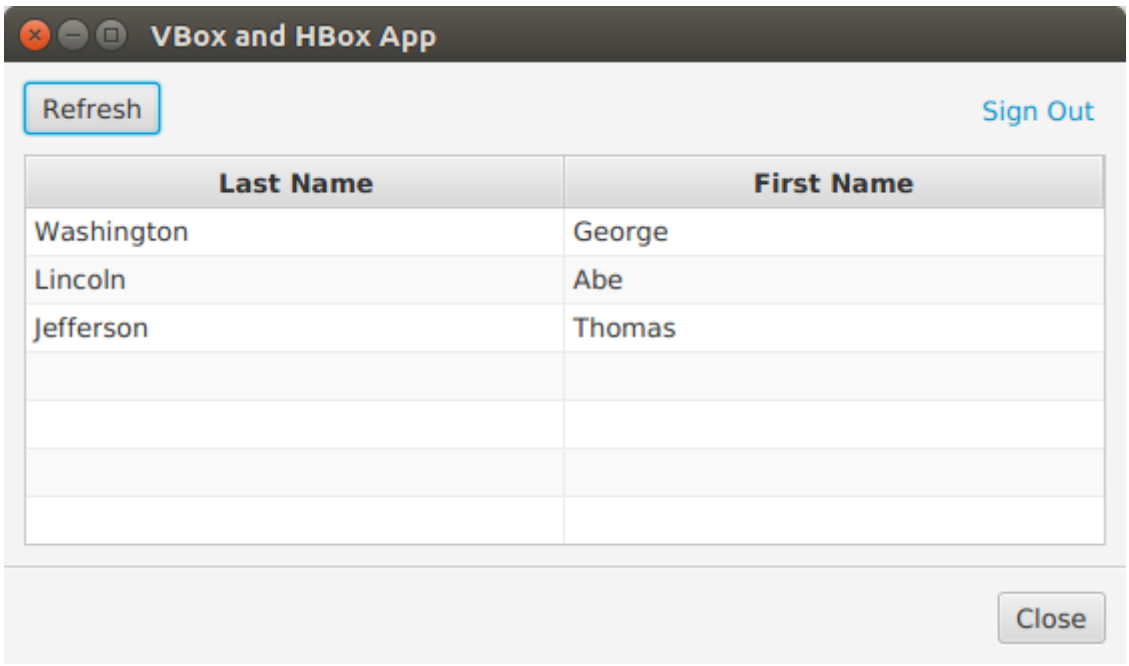
El `Insets` usado por tblCustomers omite cualquier espacio superior para mantener el espacio uniforme. JavaFX no consolida los espacios en blanco como en el diseño web. Si el Recuadro superior se estableciera en 10.0d para el `TableView`, la distancia entre los controles superiores y el `TableView` sería el doble de ancha que la distancia entre cualquiera de los otros controles.

Tenga en cuenta que estos son métodos estáticos como el `Priority`.

Esta imagen muestra la aplicación cuando se ejecuta en su tamaño inicial de 800x600.



Esta imagen muestra la aplicación redimensionada a un alto y ancho más pequeños.



4.1.5. Seleccione los contenedores correctos

La filosofía del diseño de JavaFX es la misma que la filosofía de Swing. Seleccione el contenedor adecuado para la tarea en cuestión. Este artículo presentó los dos contenedores más versátiles: `VBox` y HBox. Al establecer propiedades como alineación, hgrow y vgrow, puede crear diseños increíblemente complejos mediante el anidamiento. Estos son los contenedores que más uso y, a menudo, son los únicos contenedores que necesito.

4.1.6. Código completo

El código se puede probar en un par de archivos .java. Hay un POJO para el objeto Cliente utilizado por el `TableView`

```
public class Customer {

    private String firstName;
    private String lastName;

    public Customer(String firstName,
                    String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

`Application` Esta es la subclase JavaFX completa y principal.

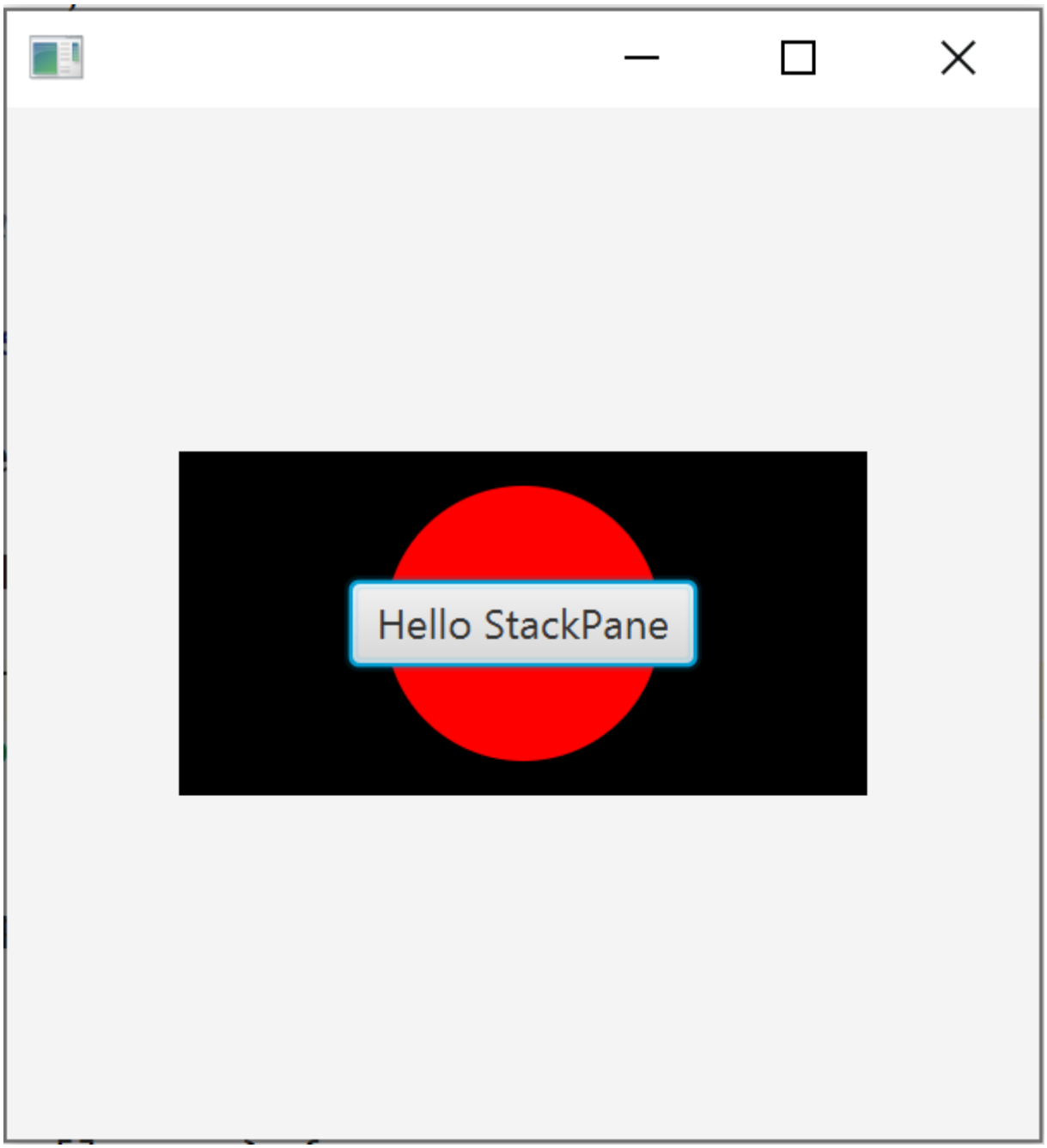
```
public class VBoxAndHBoxApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        VBox vbox = new VBox();

        HBox topControls = new HBox();
```

```
        VBox.setMargin( topControls, new Insets(10.0d) );
        topControls.setAlignment( Pos.BOTTOM_LEFT );

        Button btnRefresh = new Button("Refresh");

        HBox topRightControls = new HBox();
        HBox.setHgrow(topRightControls, Priority.ALWAYS );
        topRightControls.setAlignment( Pos.BOTTOM_RIGHT );
        Hyperlink signOutLink = new Hyperlink("Sign Out");
        topRightControls.getChildren().add( signOutLink );

        topControls.getChildren().addAll( btnRefresh, topRightControls );

        TableView<Customer> tblCustomers = new TableView<>();
        tblCustomers.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        VBox.setMargin( tblCustomers, new Insets(0.0d, 10.0d, 10.0d, 10.0d) );
        VBox.setVgrow( tblCustomers, Priority.ALWAYS );

        TableColumn<Customer, String> lastNameCol = new TableColumn<>("Last Name");
        lastNameCol.setCellValueFactory(new PropertyValueFactory<>("lastName"));

        TableColumn<Customer, String> firstNameCol = new TableColumn<>("First Name");
        firstNameCol.setCellValueFactory(new PropertyValueFactory<>("firstName"));

        tblCustomers.getColumns().addAll( lastNameCol, firstNameCol );

        Separator sep = new Separator();

        HBox bottomControls = new HBox();
        bottomControls.setAlignment(Pos.BOTTOM_RIGHT );
        VBox.setMargin( bottomControls, new Insets(10.0d) );

        Button btnClose = new Button("Close");

        bottomControls.getChildren().add( btnClose );

        vbox.getChildren().addAll(
                topControls,
                tblCustomers,
                sep,
                bottomControls
        );

        Scene scene = new Scene(vbox );

        primaryStage.setScene( scene );
        primaryStage.setWidth( 800 );
        primaryStage.setHeight( 600 );
        primaryStage.setTitle("VBox and HBox App");
        primaryStage.setOnShown( (evt) -> loadTable(tblCustomers) );
        primaryStage.show();
    }
```

```
61
62      public static void main(String[] args) {
63          launch(args);
64      }
65
66      private void loadTable(TableView<Customer> tblCustomers) {
67          tblCustomers.getItems().add(new Customer("George", "Washington"));
68          tblCustomers.getItems().add(new Customer("Abe", "Lincoln"));
69          tblCustomers.getItems().add(new Customer("Thomas", "Jefferson"));
70      }
71  }
```

4.2. StackPane

`StackPane` coloca a sus hijos uno encima de otro. El último agregado `Node` es el más alto. Por defecto `StackPane` alineará los hijos usando `Pos.CENTER`, como se puede ver en la siguiente imagen, donde están los 3 hijos (en orden de suma): `Rectangle`, `Circle` y `Button`.



Esta imagen fue producida por el siguiente fragmento:

```
1  public class StackPaneApp extends Application {
2      @Override
3      public void start(Stage stage) throws Exception {
4          StackPane pane = new StackPane(
5                  new Rectangle(200, 100, Color.BLACK),
6                  new Circle(40, Color.RED),
7                  new Button("Hello StackPane")
8          );
9
10         stage.setScene(new Scene(pane, 300, 300));
11         stage.show();
12     }
```

```
13
14      public static void main(String[] args) {
15          launch(args);
16      }
17  }
```

Podemos cambiar la alineación predeterminada agregando `pane.setAlignment(Pos.CENTER_LEFT);` para producir el siguiente efecto:

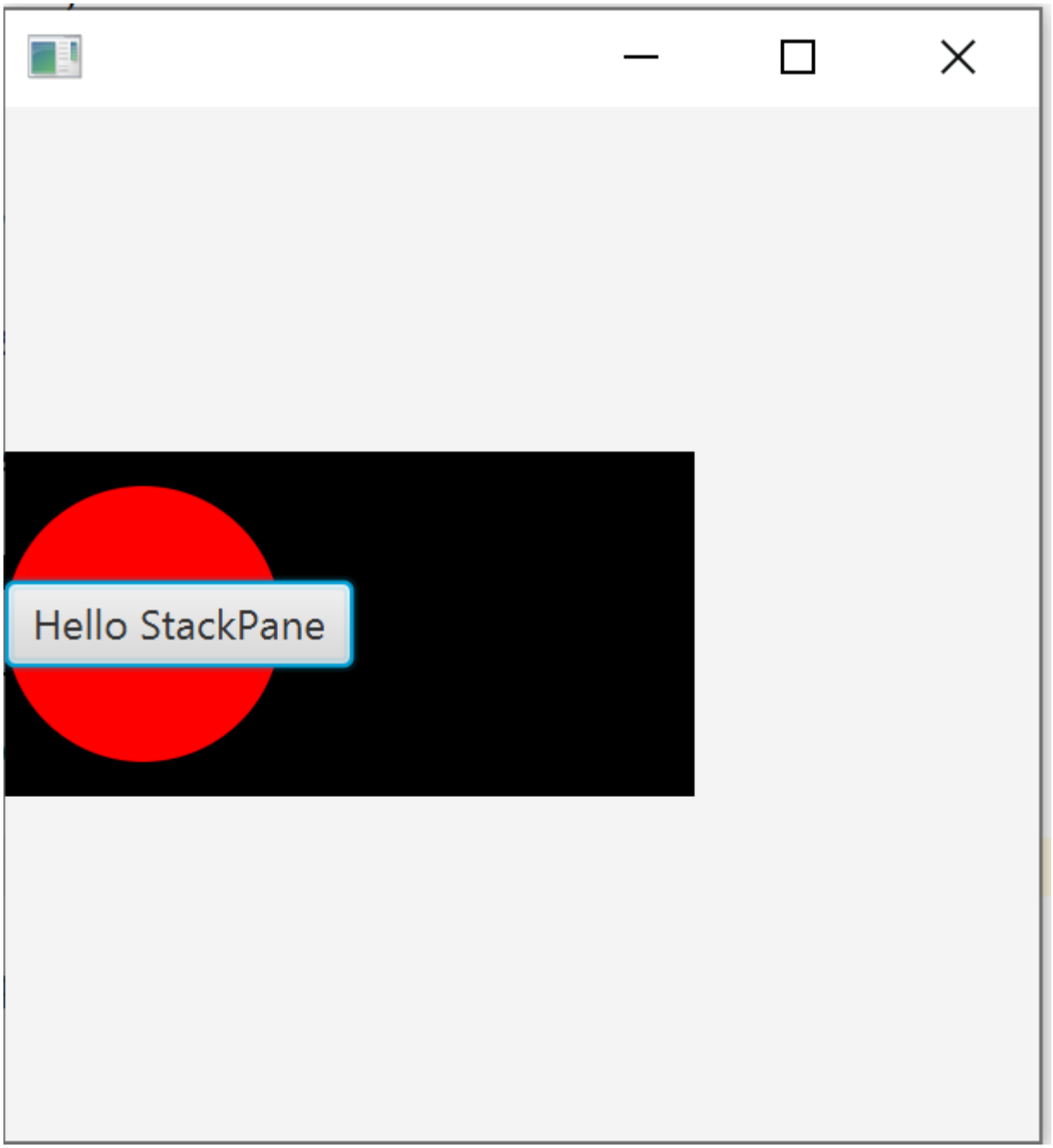


Figura 23. StackPane alineado a la izquierda

4.3. Posicionamiento absoluto con panel

Contenedores como `VBox` o `BorderPane` alinear y distribuir a sus hijos. La superclase `Pane` también es un contenedor, pero no impone un orden a sus hijos. Los hijos se posicionan a sí mismos a través de propiedades como x, centerX y layoutX. Esto se llama posicionamiento absoluto y es una técnica para colocar un `Shape` o un `Node` en un lugar determinado de la pantalla.

Esta captura de pantalla muestra una vista Acerca de. La vista Acerca de contiene `Hyperlink` en el medio de la pantalla "Acerca de esta aplicación". La vista Acerca de utiliza varias formas JavaFX para formar un diseño que se recorta para que parezca una tarjeta de presentación.

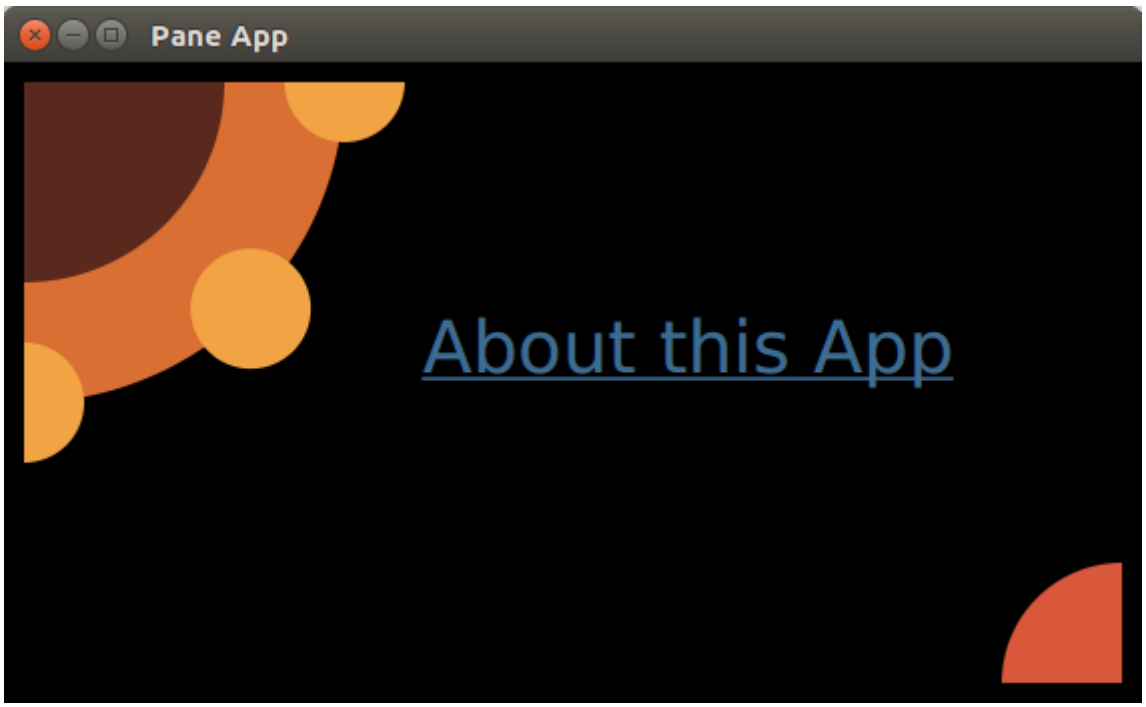


Figura 24. Captura de pantalla de la vista Acerca de en PaneApp

4.3.1. Tamaño del panel

A diferencia de la mayoría de los contenedores, `Pane` cambia de tamaño para adaptarse a su contenido y no al revés. Esta imagen es una captura de pantalla de Scenic View tomada antes de agregar el Arco inferior derecho. El `Pane` es el área resaltada en amarillo. Tenga en cuenta que no ocupa la totalidad `Stage`.

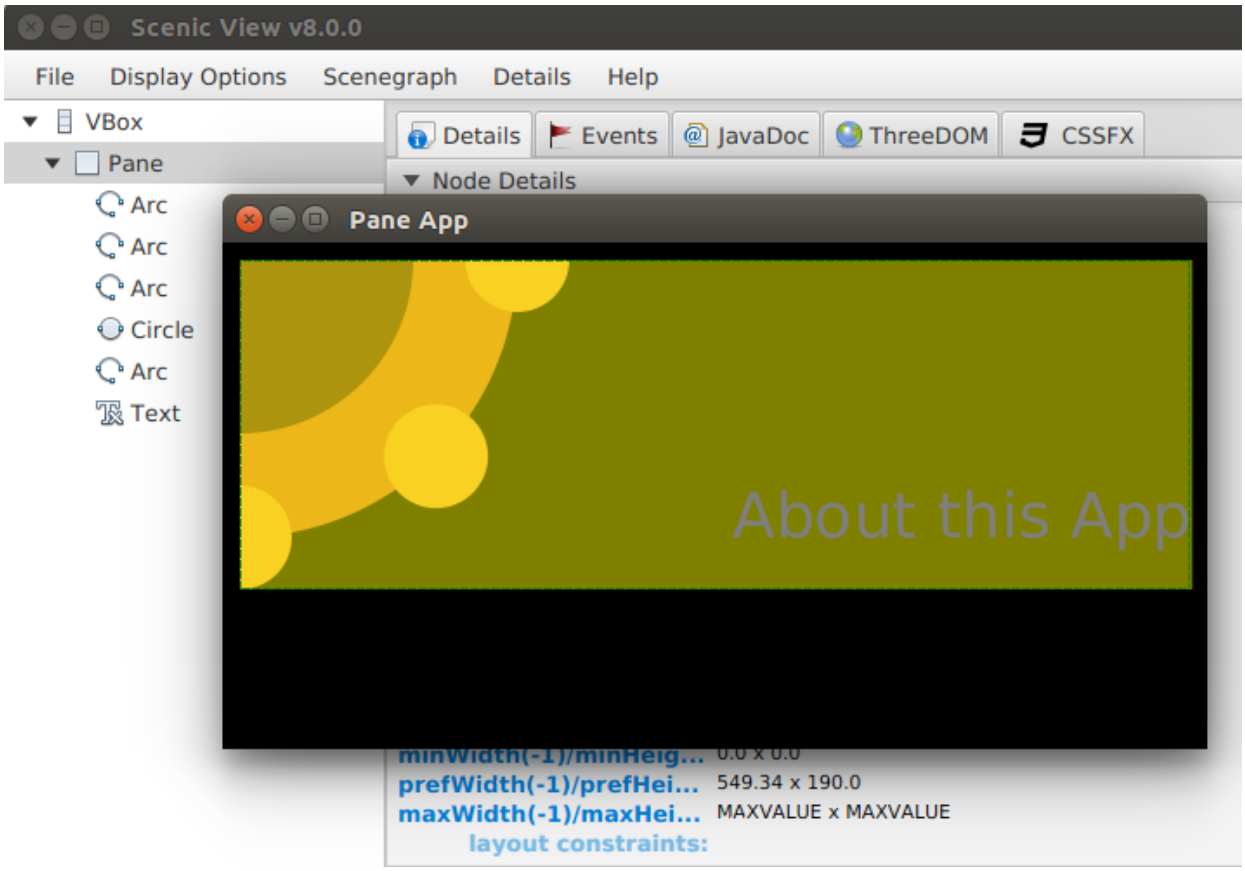


Figura 25. Vista escénica resaltando la pantalla parcialmente construida

Esta es una captura de pantalla tomada después de agregar la esquina inferior derecha `Arc`. Esto `Arc` se colocó más cerca del borde inferior derecho del archivo `Stage`. Esto obliga al Panel a estirarse para acomodar los contenidos expandidos.

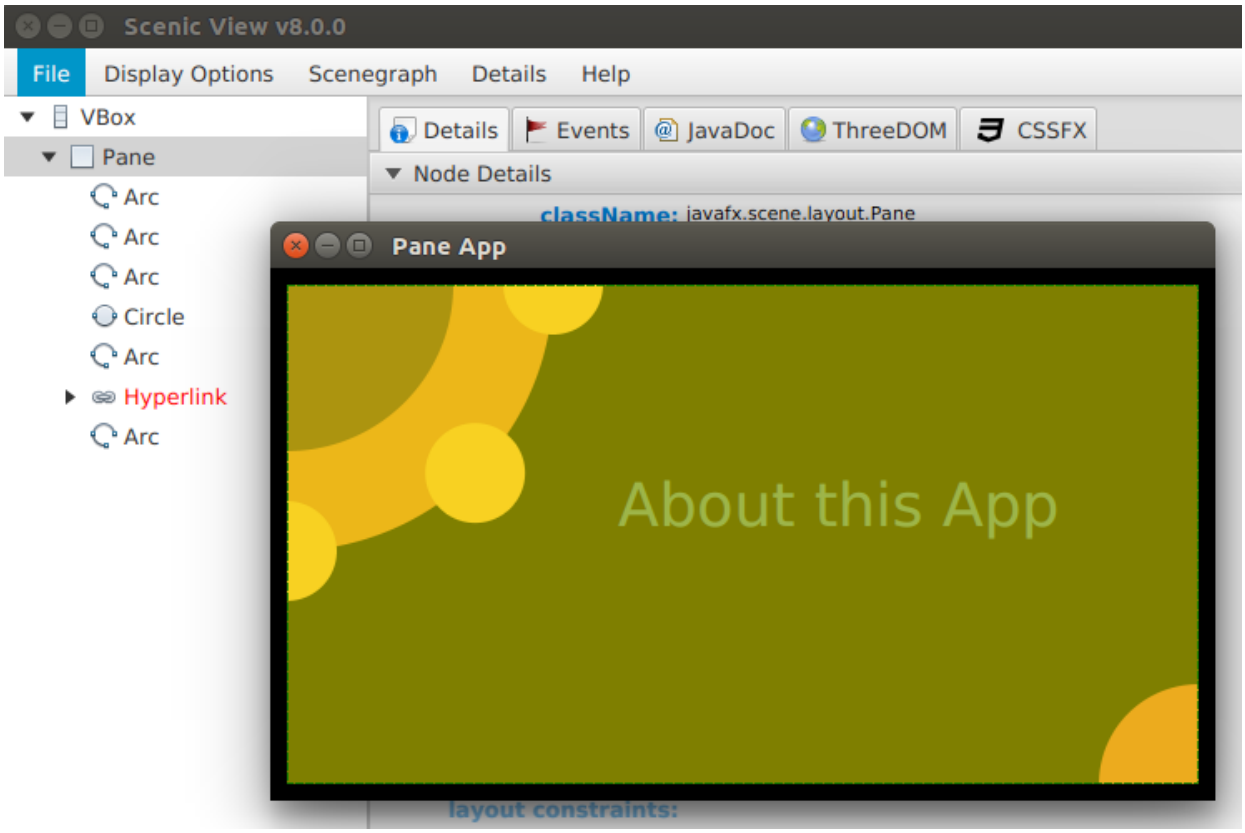


Figura 26. Vista escénica resaltando el panel expandido

4.3.2. el panel

El contenedor más externo de la vista Acerca de es un `VBox` cuyo único contenido es el archivo `Pane`. El `VBox` se utiliza para encajar en el conjunto `Stage` y proporciona un fondo.

```
VBox vbox = new VBox();
vbox.setPadding( new Insets( 10 ) );
vbox.setBackground(
    new Background(
        new BackgroundFill(Color.BLACK, new CornerRadii(0), new Insets(0))
        ));

Pane p = new Pane();
```

4.3.3. Las formas

En la parte superior izquierda de la pantalla, hay un grupo de 4 'Arcos' y 1 'Círculo'. Este código posiciona largeArc en (0,0) a través de los argumentos centerX y centerY en el `Arc` constructor. Observe que backgroundArc también se coloca en (0,0) y aparece debajo de largeArc. `Pane` no intenta eliminar el conflicto de formas superpuestas y, en este caso, lo que se busca es la superposición. smArc1 se coloca en (0,160), que está abajo en el eje Y. smArc2 está posicionado en (160,0) que está justo en el eje X. smCircle se coloca a la misma distancia que smArc1 y smArc2, pero en un ángulo de 45 grados.

```
Arc largeArc = new Arc(0, 0, 100, 100, 270, 90);
largeArc.setType(ArcType.ROUND);

Arc backgroundArc = new Arc(0, 0, 160, 160, 270, 90 );
backgroundArc.setType( ArcType.ROUND );

Arc smArc1 = new Arc( 0, 160, 30, 30, 270, 180);
smArc1.setType(ArcType.ROUND);

Circle smCircle = new Circle(160/Math.sqrt(2.0), 160/Math.sqrt(2.0),
30,Color.web("0xF2A444"));

Arc smArc2 = new Arc( 160, 0, 30, 30, 180, 180);
smArc2.setType(ArcType.ROUND);
```

La parte inferior derecha `Arc` se coloca en función de la altura total del archivo `Stage`. Los 20 restados de la altura son los 10 píxeles `Insets` de `VBox` (10 para la izquierda + 10 para la derecha).

```
Arc medArc = new Arc(568-20, 320-20, 60, 60, 90, 90);
medArc.setType(ArcType.ROUND);

primaryStage.setWidth( 568 );
primaryStage.setHeight( 320 );
```

4.3.4. El hipervínculo

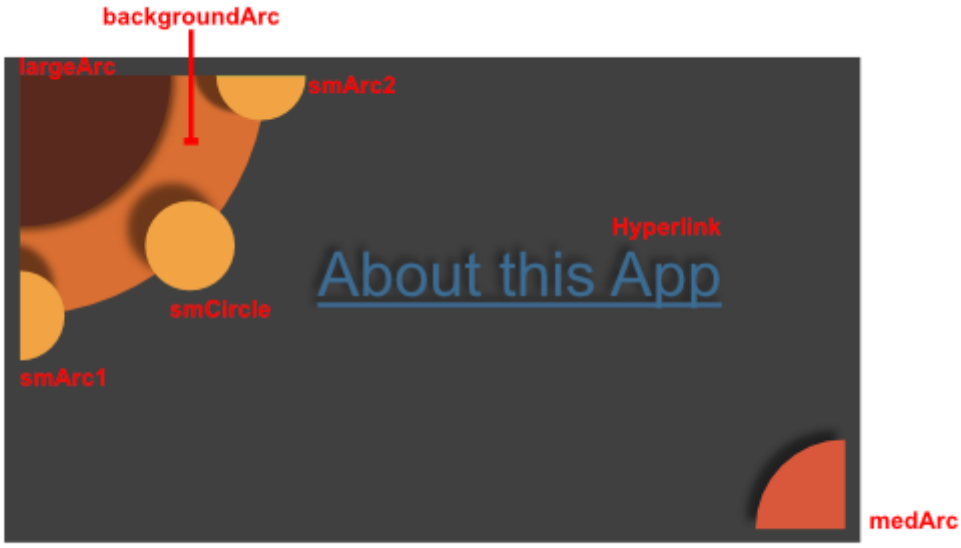
El `Hyperlink` está posicionado compensado el centro (284,160) que es el ancho y alto de `Stage` ambos dividido por dos. Esto coloca el texto del `Hyperlink` en el cuadrante inferior derecho de la pantalla, por lo que se necesita un desplazamiento basado en el `Hyperlink` ancho y el alto. Las dimensiones no están disponibles `Hyperlink` hasta que se muestra la pantalla, por lo que realizo un ajuste posterior a la visualización de la posición.

```
Hyperlink hyperlink = new Hyperlink("About this App");

primaryStage.setOnShown( (evt) -> {
     hyperlink.setLayoutX( 284 - (hyperlink.getWidth()/3) );
     hyperlink.setLayoutY( 160 - hyperlink.getHeight() );
});
```

El `Hyperlink` no está colocado en el verdadero centro de la pantalla. El valor de layoutX se basa en una operación de división por tres que lo aleja del diseño superior izquierdo.

4.3.5. Orden Z

Como se mencionó anteriormente, `Pane` admite la superposición de niños. Esta imagen muestra la vista Acerca de con profundidad añadida al diseño superior izquierdo. El más pequeño `Arcs` y `Circle` el cursor sobre backgroundArc al igual que largeArc.



El orden z en este ejemplo está determinado por el orden en que se agregan los elementos secundarios al archivo `Pane`. backgroundArc está oscurecido por elementos agregados más tarde, más notablemente largeArc. Para reorganizar los elementos secundarios, use los métodos toFront() y toBack() después de agregar los elementos al archivo `Pane`.

```
p.getChildren().addAll( backgroundArc, largeArc, smArc1, smCircle, smArc2, hyperlink,
medArc );

vbox.getChildren().add( p );
```

Al iniciar JavaFX, es tentador construir un diseño absoluto. Tenga en cuenta que los diseños absolutos son frágiles y, a menudo, se rompen cuando se cambia el tamaño de la pantalla o cuando se agregan elementos durante la fase de mantenimiento del software. Sin embargo, existen buenas razones para utilizar el posicionamiento absoluto. El juego es uno de esos usos. En un juego, puede ajustar la coordenada (x,y) de una 'Forma' para mover una pieza del juego por la pantalla. Este artículo demostró la clase JavaFX `Pane`que proporciona un posicionamiento absoluto a cualquier interfaz de usuario basada en formas.

4.3.6. Código completado

`Application` Esta es la subclase JavaFX completa y principal.

```
public class PaneApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        VBox vbox = new VBox();
        vbox.setPadding( new Insets( 10 ) );
        vbox.setBackground(
            new Background(
                new BackgroundFill(Color.BLACK, new CornerRadii(0), new Insets(0))
```

```
                ));

        Pane p = new Pane();

        Arc largeArc = new Arc(0, 0, 100, 100, 270, 90);
        largeArc.setFill(Color.web("0x59291E"));
        largeArc.setType(ArcType.ROUND);

        Arc backgroundArc = new Arc(0, 0, 160, 160, 270, 90 );
        backgroundArc.setFill( Color.web("0xD96F32") );
        backgroundArc.setType( ArcType.ROUND );

        Arc smArc1 = new Arc( 0, 160, 30, 30, 270, 180);
        smArc1.setFill(Color.web("0xF2A444"));
        smArc1.setType(ArcType.ROUND);

        Circle smCircle = new Circle(
            160/Math.sqrt(2.0), 160/Math.sqrt(2.0), 30,Color.web("0xF2A444")
            );

        Arc smArc2 = new Arc( 160, 0, 30, 30, 180, 180);
        smArc2.setFill(Color.web("0xF2A444"));
        smArc2.setType(ArcType.ROUND);

        Hyperlink hyperlink = new Hyperlink("About this App");
        hyperlink.setFont( Font.font(36) );
        hyperlink.setTextFill( Color.web("0x3E6C93") );
        hyperlink.setBorder( Border.EMPTY );

        Arc medArc = new Arc(568-20, 320-20, 60, 60, 90, 90);
        medArc.setFill(Color.web("0xD9583B"));
        medArc.setType(ArcType.ROUND);

        p.getChildren().addAll( backgroundArc, largeArc, smArc1, smCircle,
            smArc2, hyperlink, medArc );

        vbox.getChildren().add( p );

        Scene scene = new Scene(vbox);
        scene.setFill(Color.BLACK);

        primaryStage.setTitle("Pane App");
        primaryStage.setScene( scene );
        primaryStage.setWidth( 568 );
        primaryStage.setHeight( 320 );
        primaryStage.setOnShown( (evt) -> {
             hyperlink.setLayoutX( 284 - (hyperlink.getWidth()/3) );
             hyperlink.setLayoutY( 160 - hyperlink.getHeight() );
        });
        primaryStage.show();
    }

```

```
63      public static void main(String[] args) {
64          launch(args);
65      }
66  }
```

4.4. Recorte

La mayoría de los contenedores de diseño JavaFX (clase base Región) posicionan y dimensionan automáticamente a sus elementos secundarios, por lo que recortar cualquier contenido secundario que pueda sobresalir más allá de los límites del diseño del contenedor nunca es un problema. La gran excepción es Pane , una subclase directa `Region` y la clase base para todos los contenedores de diseño con elementos secundarios de acceso público. A diferencia de sus subclases, Pane no intenta organizar a sus hijos, sino que simplemente acepta el posicionamiento y el tamaño explícitos del usuario.

Esto lo hace `Pane` adecuado como una superficie de dibujo, similar a Canvas, pero representa elementos secundarios de Forma definidos por el usuario en lugar de comandos de dibujo directos. El problema es que normalmente se espera que las superficies de dibujo recorten automáticamente su contenido en sus límites. `Canvas` hace esto por defecto pero `Pane` no lo hace. Desde el último párrafo de la entrada de Javadoc para `Pane`:

> El panel no recorta su contenido de forma predeterminada, por lo que es posible que los límites de los elementos secundarios se extiendan más allá de sus propios límites, ya sea si los elementos secundarios se colocan en coordenadas negativas o si el panel se redimensiona más pequeño que su tamaño preferido.

Esta cita es algo engañosa. Los elementos secundarios se representan (total o parcialmente) fuera de su elemento principal `Pane` 'siempre que' su combinación de posición y tamaño se extienda más allá de los límites del elemento principal, independientemente de si la posición es negativa o si `Pane` alguna vez se redimensiona. En pocas palabras, `Pane` solo proporciona un cambio de coordenadas a sus elementos secundarios, en función de su esquina superior izquierda, pero sus límites de diseño se ignoran por completo al representar elementos secundarios. Tenga en cuenta que el Javadoc para todas las `Pane` subclases (que revisé) incluye una advertencia similar. Tampoco recortan su contenido, pero como se mencionó anteriormente, esto no suele ser un problema para ellos porque organizan automáticamente a sus hijos.

Entonces, para usarlo correctamente `Pane` como superficie de dibujo para `Shapes`, necesitamos recortar manualmente su contenido. Esto es algo complejo, especialmente cuando se trata de un borde visible. Escribí una pequeña aplicación de demostración para ilustrar el comportamiento predeterminado y varios pasos para solucionarlo. Puede descargarlo como PaneDemo.zip que contiene un proyecto para NetBeans 8.2 y Java SE 8u112. Las siguientes secciones explican cada paso con capturas de pantalla y fragmentos de código pertinentes.

4.4.1. Comportamiento por defecto

Al comenzar, PaneDemo muestra lo que sucede cuando coloca una `Ellipse` forma en un espacio `Pane` que es demasiado pequeño para contenerla por completo. `Pane` Tiene un bonito borde grueso y redondeado para visualizar su área . La ventana de la aplicación es redimensionable, con el `Pane` tamaño siguiendo el tamaño de la ventana. Los tres botones de la izquierda se utilizan para

cambiar a los otros pasos de la demostración; haga clic en Predeterminado (Alt+D) para volver a la salida predeterminada de un paso posterior.

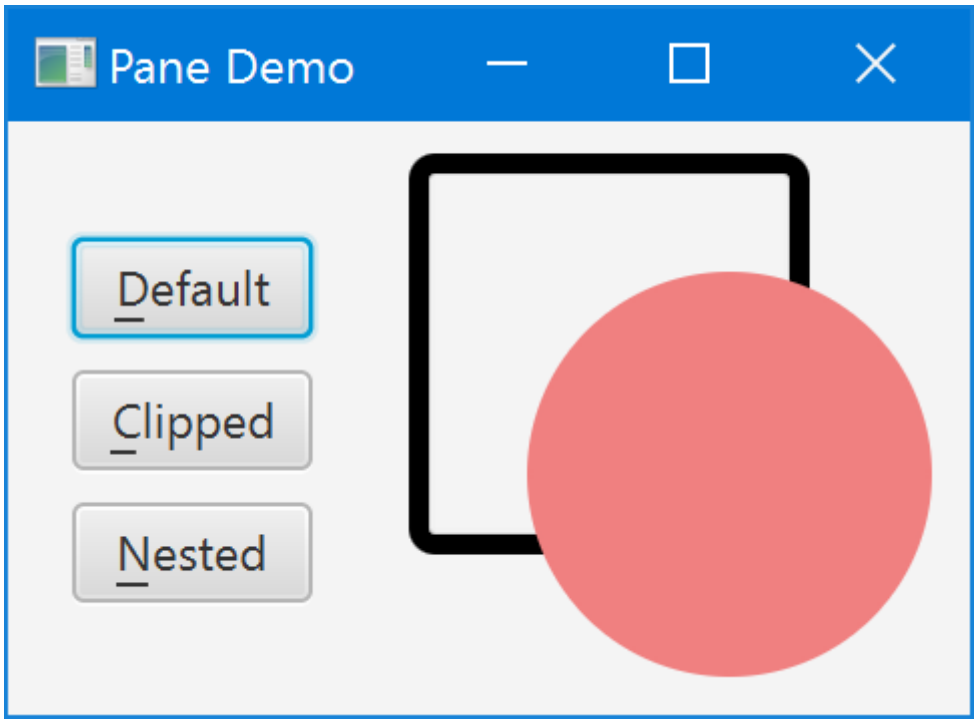


Figura 28. Niño que se extiende fuera de los límites del panel

Como puede ver, el `Ellipse` sobrescribe el de su padre `Border` y sobresale mucho más allá. El siguiente código se utiliza para generar la vista predeterminada. Se divide en varios métodos más pequeños y una constante para el `Border` radio de la esquina, ya que se hará referencia a ellos en los siguientes pasos.

```
static final double BORDER_RADIUS = 4;

static Border createBorder() {
    return new Border(
            new BorderStroke(Color.BLACK, BorderStrokeStyle.SOLID,
            new CornerRadii(BORDER_RADIUS), BorderStroke.THICK));
}

static Shape createShape() {
    final Ellipse shape = new Ellipse(50, 50);
    shape.setCenterX(80);
    shape.setCenterY(80);
    shape.setFill(Color.LIGHTCORAL);
    shape.setStroke(Color.LIGHTCORAL);
    return shape;
}

static Region createDefault() {
    final Pane pane = new Pane(createShape());
    pane.setBorder(createBorder());
    pane.setPrefSize(100, 100);
    return pane;
```

```
}
```

4.4.2. Recorte simple

Sorprendentemente, no hay una opción predefinida para hacer que un redimensionable `Region` recorte automáticamente a sus elementos secundarios a su tamaño actual. En su lugar, debe usar la propiedad clipProperty básica definida en `Node` y mantenerla actualizada manualmente para reflejar los cambios en los límites del diseño. El método `clipChildren` a continuación muestra cómo funciona esto (con Javadoc porque es posible que desee reutilizarlo en su propio código):

```java
/**
 * Clips the children of the specified {@link Region} to its current size.
 * This requires attaching a change listener to the region's layout bounds,
 * as JavaFX does not currently provide any built-in way to clip children.
 *
 * @param region the {@link Region} whose children to clip
 * @param arc the {@link Rectangle#arcWidth} and {@link Rectangle#arcHeight}
 *            of the clipping {@link Rectangle}
 * @throws NullPointerException if {@code region} is {@code null}
 */
static void clipChildren(Region region, double arc) {

    final Rectangle outputClip = new Rectangle();
    outputClip.setArcWidth(arc);
    outputClip.setArcHeight(arc);
    region.setClip(outputClip);

    region.layoutBoundsProperty().addListener((ov, oldValue, newValue) -> {
        outputClip.setWidth(newValue.getWidth());
        outputClip.setHeight(newValue.getHeight());
    });
}

static Region createClipped() {
    final Pane pane = new Pane(createShape());
    pane.setBorder(createBorder());
    pane.setPrefSize(100, 100);

    // clipped children still overwrite Border!
    clipChildren(pane, 3 * BORDER_RADIUS);

    return pane;
}
```

Elija Recortado (Alt+C) en PaneDemo para representar la salida correspondiente. Así es como se ve:

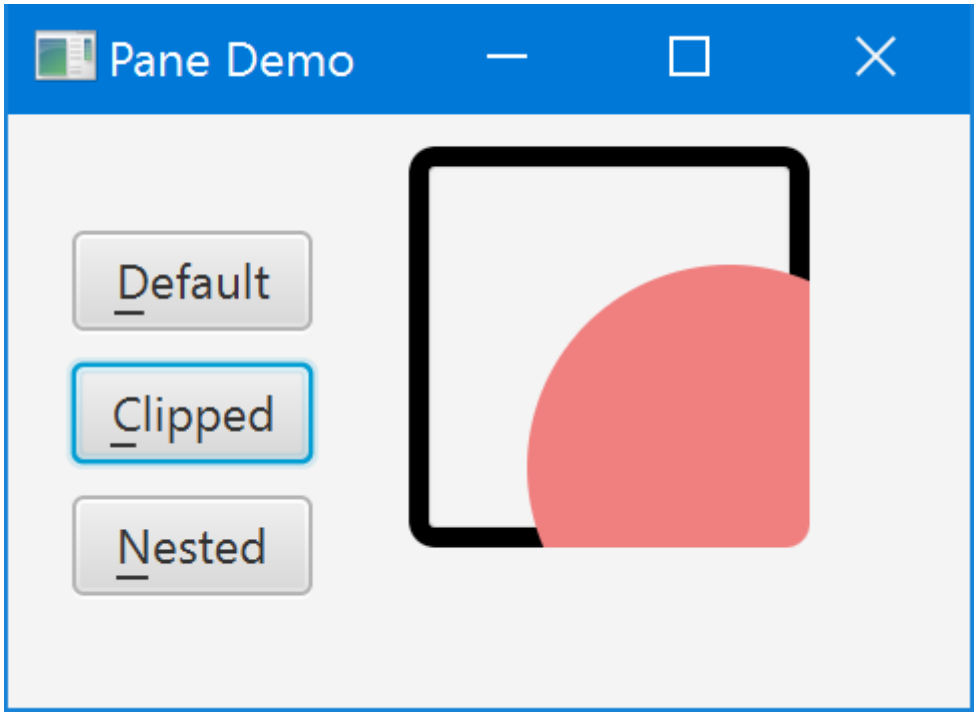


Figura 29. Panel con clip aplicado

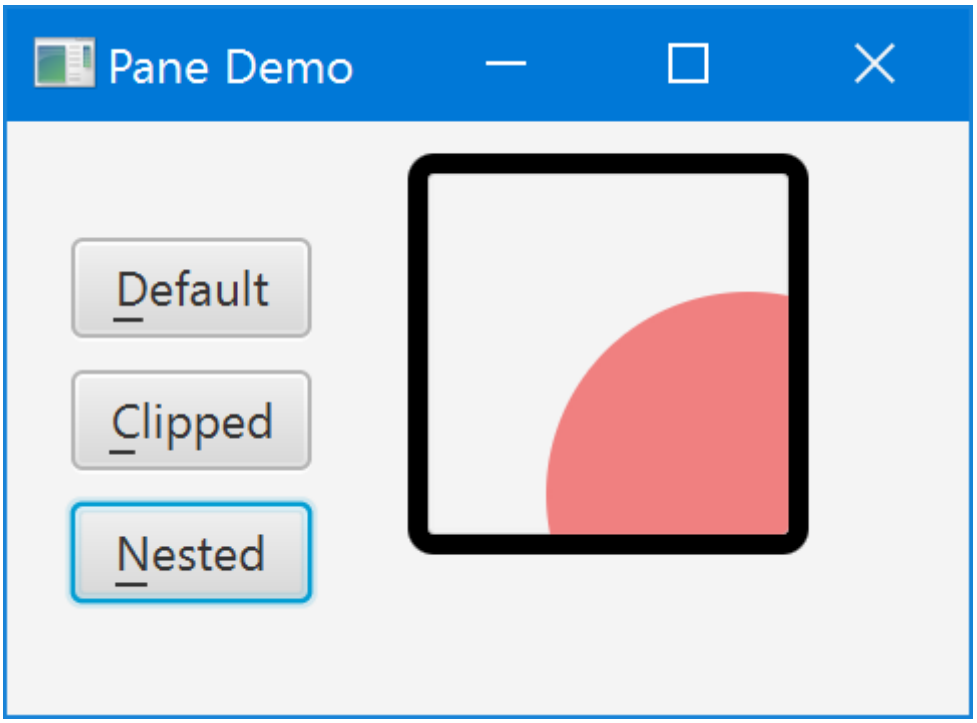
Eso es mejor. El `Ellipse` ya no sobresale más allá del `Pane` – pero todavía sobrescribe su Borde. También tenga en cuenta que tuvimos que especificar manualmente un redondeo de esquina estimado para el recorte `Rectangle` para reflejar las `Border` esquinas redondeadas. Esta estimación es 3 * BORDER_RADIUS porque el radio de esquina especificado en `Border` realmente define su radio interior, y el radio exterior (que necesitamos aquí) será mayor dependiendo del `Border` grosor. (Podría calcular el radio exterior exactamente si realmente quisiera, pero lo omití para la aplicación de demostración).

4.4.2.1. 4.4.3. Paneles anidados

¿Podemos de alguna manera especificar una región de recorte que excluya un 'Borde' visible? No en el dibujo `Pane` en sí, que yo sepa. La región de recorte afecta `Border` tanto al contenido como a otros, por lo que si tuviera que reducir la región de recorte para excluirla, ya no vería nada `Border`. En su lugar, la solución es crear dos paneles anidados: un dibujo interior `Pane` sin `Border` que se ajuste exactamente a sus límites y otro exterior `StackPane` que defina lo visible `Border` y también cambie el tamaño del dibujo `Pane`. Aquí está el código final:

```
static Region createNested() {
    // create drawing Pane without Border or size
    final Pane pane = new Pane(createShape());
    clipChildren(pane, BORDER_RADIUS);

    // create sized enclosing Region with Border
    final Region container = new StackPane(pane);
    container.setBorder(createBorder());
    container.setPrefSize(100, 100);
    return container;
}
```

Elija Anidado (Alt+N) en PaneDemo para representar la salida correspondiente. Ahora todo se ve como debería:



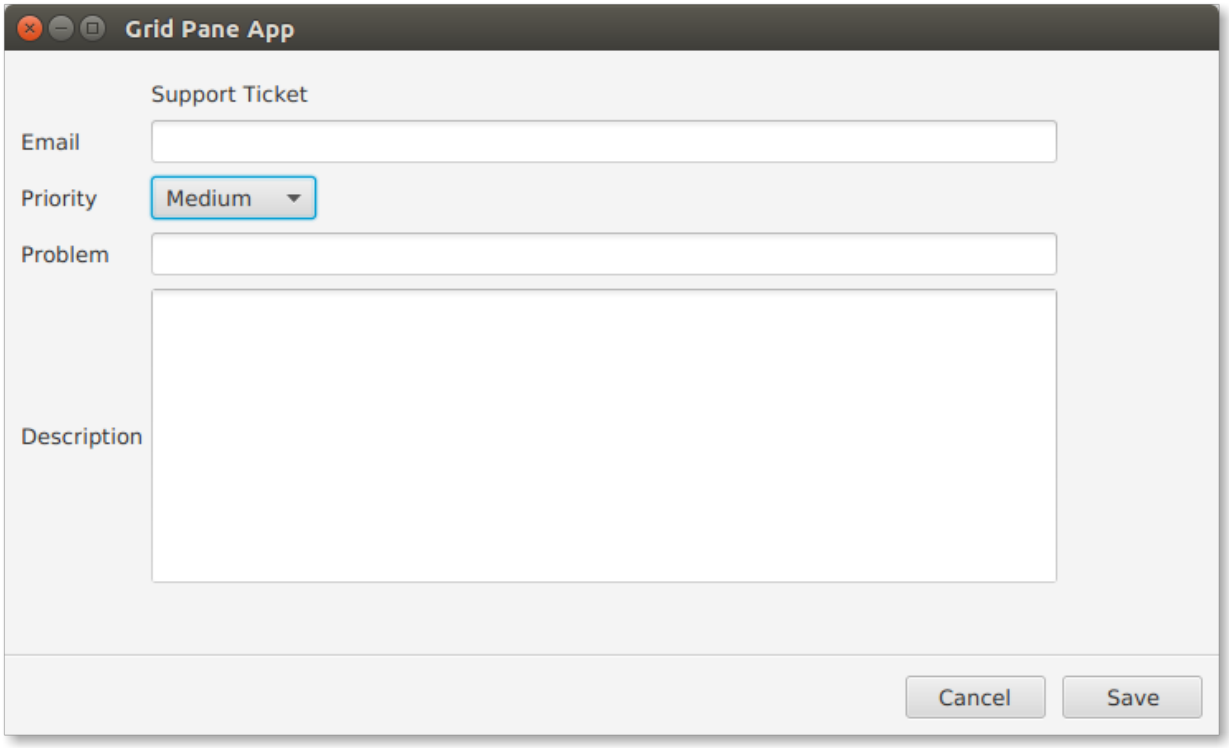
Como beneficio adicional, ya no necesitamos estimar un radio de esquina correcto para el recorte `Rectangle`. Ahora recortamos la circunferencia interior en lugar de la exterior de nuestro visible `Border`, para que podamos reutilizar directamente su radio de esquina interior. Si especifica múltiples radios de esquina diferentes o uno más complejo `Border`, tendrá que definir un recorte correspondientemente más complejo `Shape`.

Hay una pequeña advertencia. La esquina superior izquierda del dibujo `Pane` con respecto a todas las coordenadas secundarias ahora comienza *dentro* del visible `Border`. Si cambia retroactivamente uno `Pane` con paneles visibles `Border` a anidados como se muestra aquí, todos los niños exhibirán un ligero cambio de posición correspondiente al `Border` grosor.

4.5. GridPane

Los formularios en las aplicaciones comerciales a menudo usan un diseño que imita un registro de base de datos. Para cada columna de una tabla, se agrega un encabezado en el lado izquierdo que coincide con un valor de fila en el lado derecho. JavaFX tiene un control de propósito especial llamado `GridPane` para este tipo de diseño que mantiene los contenidos alineados por fila y columna. `GridPane` también admite expansión para diseños más complejos.

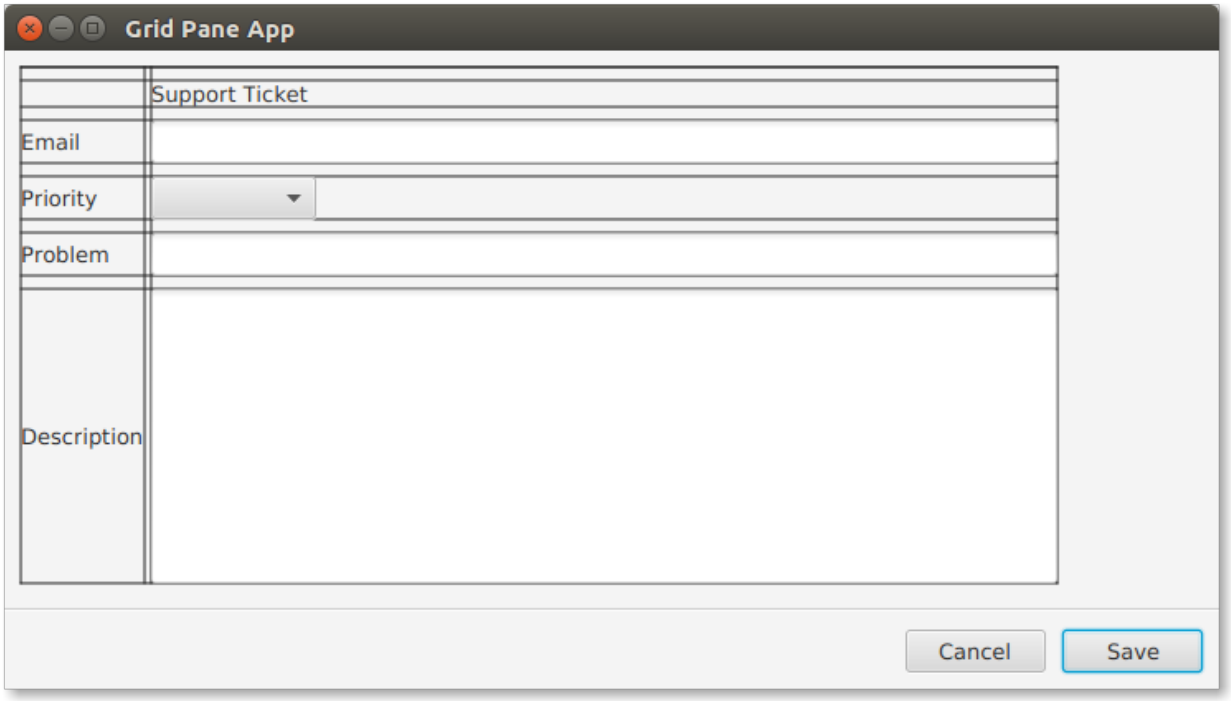
`GridPane` Esta captura de pantalla muestra un diseño básico . En el lado izquierdo del formulario, hay una columna de nombres de campo: Correo electrónico, Prioridad, Problema, Descripción. En el lado derecho del formulario, hay una columna de controles que mostrará el valor del campo correspondiente. Los nombres de campo son de tipo `Label` y los controles de valor son una mezcla que incluye `TextField`, `TextArea` y `ComboBox`.



El siguiente código muestra los objetos creados para el formulario. "vbox" es la raíz del `Scene` y también contendrá el `ButtonBar` en la base del formulario.

```
VBox vbox = new VBox();

GridPane gp = new GridPane();

Label lblTitle = new Label("Support Ticket");

Label lblEmail = new Label("Email");
TextField tfEmail = new TextField();

Label lblPriority = new Label("Priority");
ObservableList<String> priorities = FXCollections.observableArrayList("Medium", "High",
"Low");
ComboBox<String> cbPriority = new ComboBox<>(priorities);

Label lblProblem = new Label("Problem");
TextField tfProblem = new TextField();

Label lblDescription = new Label("Description");
TextArea taDescription = new TextArea();
```

GridPane tiene un método útil `setGridLinesVisible()` que muestra la estructura de la cuadrícula y los canalones. Es especialmente útil en diseños más complejos donde se involucra la expansión porque los espacios en las asignaciones de filas/columnas pueden causar cambios en el diseño.



4.5.1. Espaciado

Como contenedor, `GridPane` tiene una propiedad de relleno que se puede configurar para rodear el `GridPane` contenido con espacios en blanco. "relleno" tomará un `Inset` objeto como parámetro. En este ejemplo, se aplican 10 píxeles de espacio en blanco a todos los lados, por lo que se usa un constructor de formato corto para `Inset`.

Dentro de `GridPane`, vgap y hgap controlan los canalones. El hgap se establece en 4 para mantener los campos cerca de sus valores. vgap es un poco más grande para ayudar con la navegación del mouse.

```
gp.setPadding( new Insets(10) );
gp.setHgap( 4 );
gp.setVgap( 8 );
```

Para mantener consistente la parte inferior del formulario, `Priority` se establece a en el VBox. Sin embargo , esto *no cambiará el tamaño* de las filas individuales. Para especificaciones de cambio de tamaño individuales, use `ColumnConstraints` y `RowConstraints`.

```
VBox.setVgrow(gp, Priority.ALWAYS );
```

4.5.2. Adición de elementos

A diferencia de los contenedores como `BorderPane` o `HBox`, los nodos deben especificar su posición dentro del archivo `GridPane`. Esto se hace con el `add()` método en `GridPane` y no con el método add en una propiedad secundaria del contenedor. Esta forma del `GridPane` `add()` método toma una posición de columna de base cero y una posición de fila de base cero. Este código pone dos declaraciones en la misma línea para facilitar la lectura.

```
gp.add( lblTitle,       1, 1);  // empty item at 0,0
gp.add( lblEmail,       0, 2); gp.add(tfEmail,        1, 2);
gp.add( lblPriority,    0, 3); gp.add( cbPriority,    1, 3);
gp.add( lblProblem,     0, 4); gp.add( tfProblem,     1, 4);
gp.add( lblDescription, 0, 5); gp.add( taDescription, 1, 5);
```

lblTitle se coloca en la segunda columna de la primera fila. No hay ninguna entrada en la primera columna de la primera fila.

Las adiciones posteriores se presentan por parejas. Los objetos de nombre de campo `Label` se colocan en la primera columna (índice de columna=0) y los controles de valor se colocan en la segunda columna (índice de columna=1). Las filas se agregan por el segundo valor incrementado. Por ejemplo, lblPriority se coloca en la cuarta fila junto con su `ComboBox`.

`GridPane` es un contenedor importante en el diseño de aplicaciones empresariales JavaFX. Cuando tenga un requisito de pares de nombre/valor, `GridPane` será una manera fácil de admitir la fuerte orientación de columna de un formulario tradicional.

4.5.3. Código completado

La siguiente clase es el código completo del ejemplo. Esto incluye la definición de la `ButtonBar` que no se presentó en las secciones anteriores enfocadas en `GridPane`.

```
public class GridPaneApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        VBox vbox = new VBox();

        GridPane gp = new GridPane();
        gp.setPadding( new Insets(10) );
        gp.setHgap( 4 );
        gp.setVgap( 8 );

        VBox.setVgrow(gp, Priority.ALWAYS );

        Label lblTitle = new Label("Support Ticket");

        Label lblEmail = new Label("Email");
        TextField tfEmail = new TextField();

        Label lblPriority = new Label("Priority");
        ObservableList<String> priorities =
            FXCollections.observableArrayList("Medium", "High", "Low");
        ComboBox<String> cbPriority = new ComboBox<>(priorities);

        Label lblProblem = new Label("Problem");
        TextField tfProblem = new TextField();

        Label lblDescription = new Label("Description");
```

```
        TextArea taDescription = new TextArea();

        gp.add( lblTitle,       1, 1);  // empty item at 0,0
        gp.add( lblEmail,       0, 2); gp.add(tfEmail,        1, 2);
        gp.add( lblPriority,    0, 3); gp.add( cbPriority,    1, 3);
        gp.add( lblProblem,     0, 4); gp.add( tfProblem,     1, 4);
        gp.add( lblDescription, 0, 5); gp.add( taDescription, 1, 5);

        Separator sep = new Separator(); // hr

        ButtonBar buttonBar = new ButtonBar();
        buttonBar.setPadding( new Insets(10) );

        Button saveButton = new Button("Save");
        Button cancelButton = new Button("Cancel");

        buttonBar.setButtonData(saveButton, ButtonBar.ButtonData.OK_DONE);
        buttonBar.setButtonData(cancelButton, ButtonBar.ButtonData.CANCEL_CLOSE);

        buttonBar.getButtons().addAll(saveButton, cancelButton);

        vbox.getChildren().addAll( gp, sep, buttonBar );

        Scene scene = new Scene(vbox);

        primaryStage.setTitle("Grid Pane App");
        primaryStage.setScene(scene);
        primaryStage.setWidth( 736 );
        primaryStage.setHeight( 414  );
        primaryStage.show();

    }

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

4.6. GridPane Spanning

Para formularios más complejos implementados con `GridPane`, se admite la expansión. La expansión permite que un control reclame el espacio de columnas vecinas (colspan) y filas vecinas (rowspan). Esta captura de pantalla muestra un formulario que amplía el ejemplo de la sección anterior. El diseño de dos columnas de la versión anterior se reemplazó por un diseño de varias columnas. Los campos como Problema y Descripción conservan la estructura original. Pero se agregaron controles a las filas que anteriormente contenían solo Correo electrónico y Prioridad.

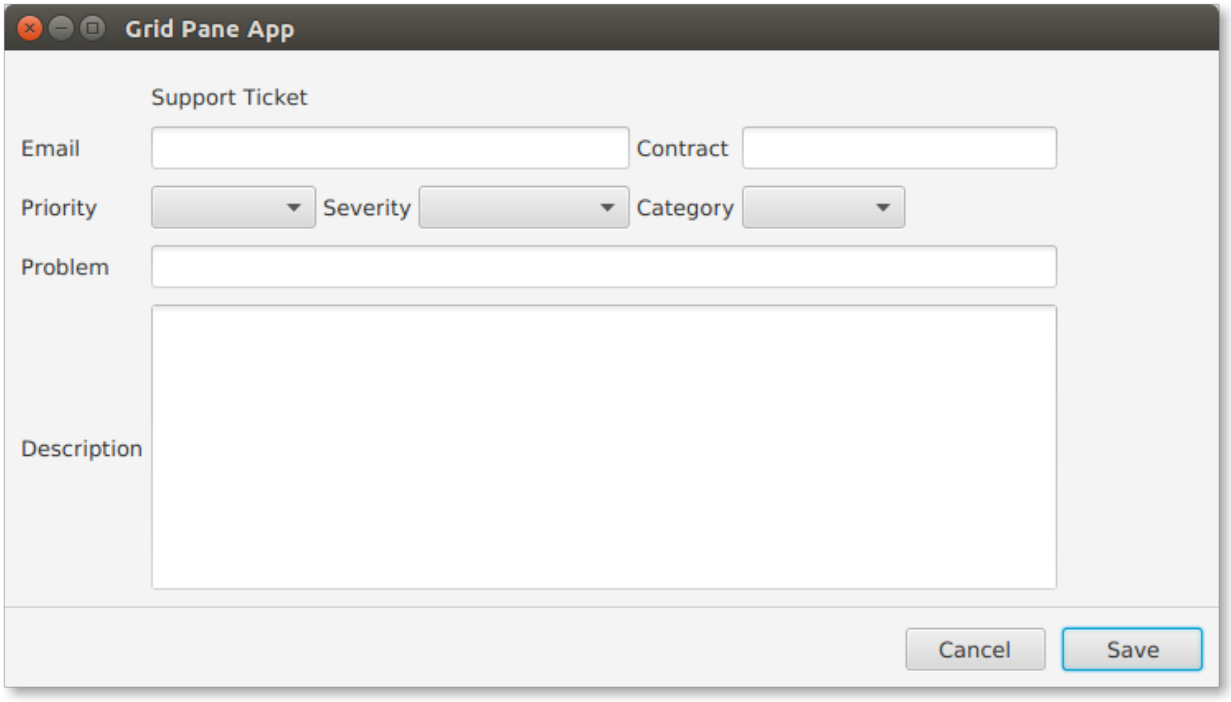


Figura 33. Columnas de expansión

Al activar las líneas de la cuadrícula, observe que la cuadrícula anterior de dos columnas se reemplaza con una cuadrícula de seis columnas. La tercera fila que contiene seis elementos (3 pares de nombre de campo/valor) dicta la estructura. El resto del formulario utilizará la expansión para completar el espacio en blanco.

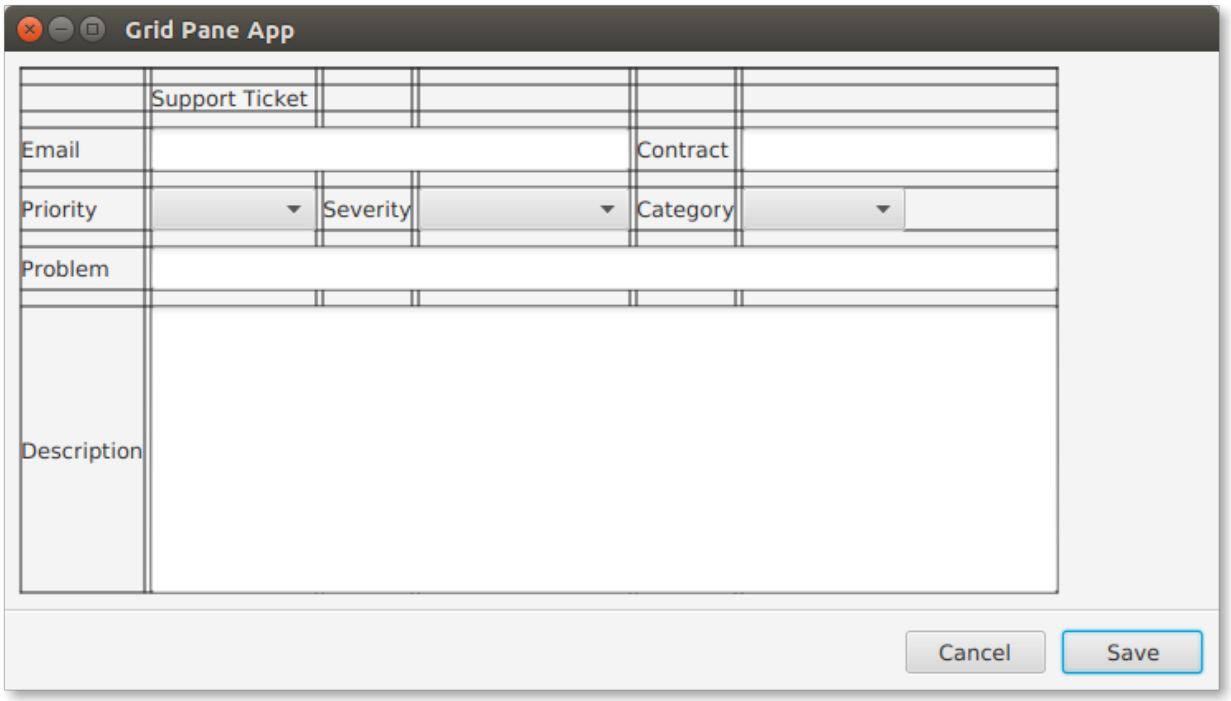


Figura 34. Líneas que resaltan la extensión

A continuación se muestran los objetos contenedor `VBox` y `GridPane` utilizados en esta actualización. Hay un poco más de Vgap para ayudar al usuario a seleccionar los `ComboBox` controles.

```
GridPane gp = new GridPane();
gp.setPadding( new Insets(10) );
gp.setHgap( 4 );
gp.setVgap( 10 );

VBox.setVgrow(gp, Priority.ALWAYS );
```

Estas son declaraciones de creación de control del ejemplo actualizado.

```
Label lblTitle = new Label("Support Ticket");

Label lblEmail = new Label("Email");
TextField tfEmail = new TextField();

Label lblContract = new Label("Contract");
TextField tfContract = new TextField();

Label lblPriority = new Label("Priority");
ObservableList<String> priorities =
    FXCollections.observableArrayList("Medium", "High", "Low");
ComboBox<String> cbPriority = new ComboBox<>(priorities);

Label lblSeverity = new Label("Severity");
ObservableList<String> severities =
    FXCollections.observableArrayList("Blocker", "Workaround", "N/A");
ComboBox<String> cbSeverity = new ComboBox<>(severities);

Label lblCategory = new Label("Category");
ObservableList<String> categories =
    FXCollections.observableArrayList("Bug", "Feature");
ComboBox<String> cbCategory = new ComboBox<>(categories);

Label lblProblem = new Label("Problem");
TextField tfProblem = new TextField();

Label lblDescription = new Label("Description");
TextArea taDescription = new TextArea();
```

Como en la versión anterior, los controles se agregan al `GridPane` método `add()`. Se especifica una columna y una fila. En este fragmento, la indexación no es sencilla, ya que se espera que se llenen los vacíos mediante el contenido expandido.

```
gp.add( lblTitle,       1, 0);  // empty item at 0,0

gp.add( lblEmail,       0, 1);
gp.add(tfEmail,         1, 1);
gp.add( lblContract,    4, 1 );
gp.add( tfContract,     5, 1 );

gp.add( lblPriority,    0, 2);
```

```
gp.add( cbPriority,     1, 2);
gp.add( lblSeverity,    2, 2);
gp.add( cbSeverity,     3, 2);
gp.add( lblCategory,    4, 2);
gp.add( cbCategory,     5, 2);

gp.add( lblProblem,     0, 3); gp.add( tfProblem,     1, 3);
gp.add( lblDescription, 0, 4); gp.add( taDescription, 1, 4);
```

Finalmente, las definiciones de expansión se establecen mediante un método estático en `GridPane`. Hay un método similar para hacer la expansión de filas. El título ocupará 5 columnas, al igual que el problema y la descripción. El correo electrónico comparte una fila con el contrato, pero ocupará más columnas. La tercera fila de ComboBoxes es un conjunto de tres pares de campo/valor, cada uno de los cuales ocupa una columna.

```
GridPane.setColumnSpan( lblTitle, 5 );
GridPane.setColumnSpan( tfEmail, 3 );
GridPane.setColumnSpan( tfProblem, 5 );
GridPane.setColumnSpan( taDescription, 5 );
```

Alternativamente, una variación del método add() tendrá argumentos columnSpan y rowSpan para evitar la subsiguiente llamada al método estático.

Este ejemplo ampliado `GridPane` demostró la expansión de columnas. La misma capacidad está disponible para la expansión de filas, lo que permitiría que un control reclame espacio vertical adicional. La expansión mantiene los controles alineados incluso en los casos en que varía el número de elementos en una fila (o columna) determinada. Para mantener el enfoque en el tema de expansión, esta cuadrícula permitió que variaran los anchos de las columnas. El artículo sobre `ColumnConstraints` y `RowConstraints` se centrará en la construcción de verdaderas cuadrículas tipográficas modulares y de columnas mediante un mejor control de las columnas (y las filas).

4.6.1. Código completado

El siguiente es el código completo para el ejemplo de GridPane de expansión.

```
public class ComplexGridPaneApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        VBox vbox = new VBox();

        GridPane gp = new GridPane();
        gp.setPadding( new Insets(10) );
        gp.setHgap( 4 );
        gp.setVgap( 10 );

        VBox.setVgrow(gp, Priority.ALWAYS );

        Label lblTitle = new Label("Support Ticket");
```

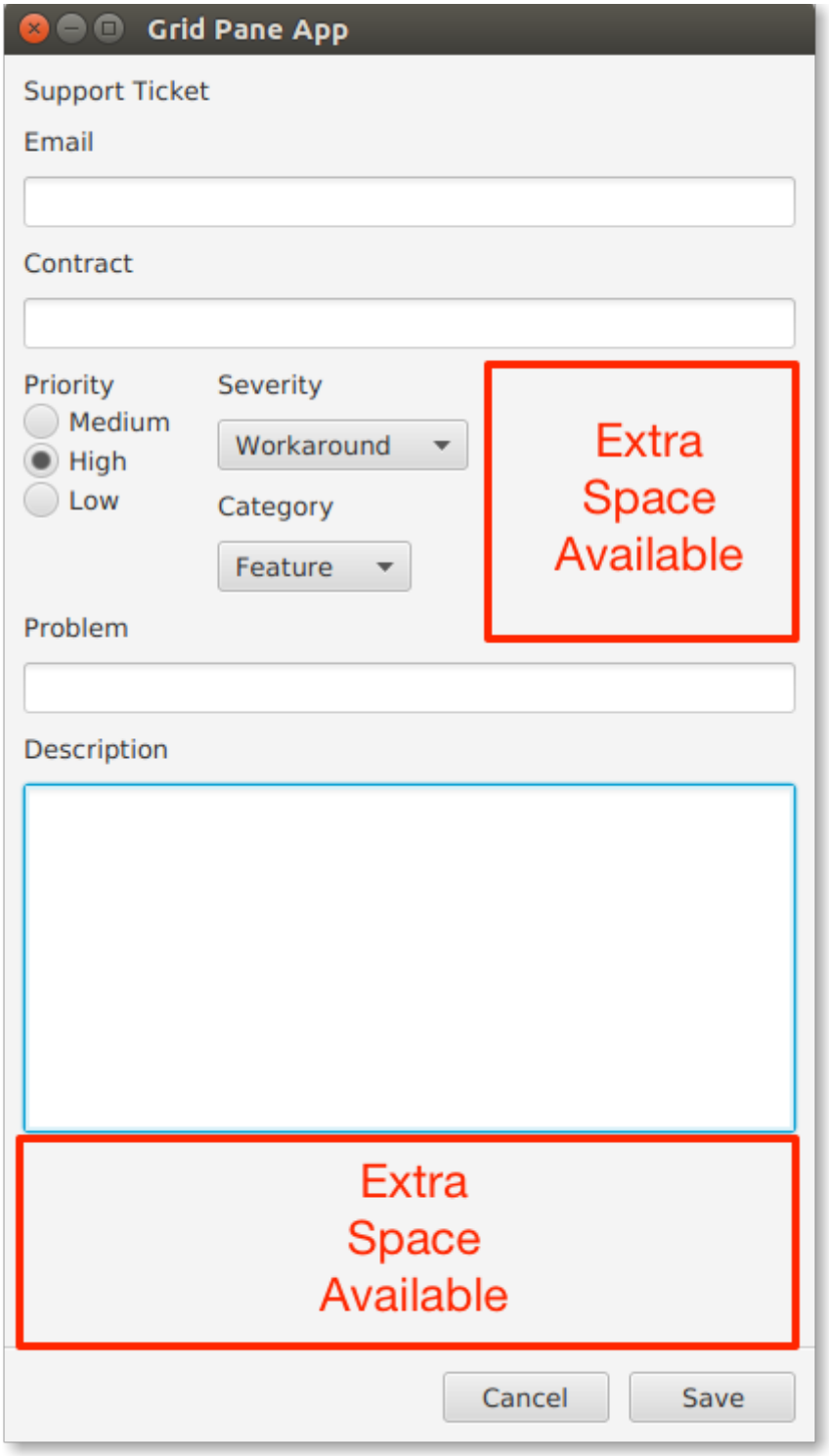
```
16
17          Label lblEmail = new Label("Email");
18          TextField tfEmail = new TextField();
19
20          Label lblContract = new Label("Contract");
21          TextField tfContract = new TextField();
22
23          Label lblPriority = new Label("Priority");
24          ObservableList<String> priorities =
25              FXCollections.observableArrayList("Medium", "High", "Low");
26          ComboBox<String> cbPriority = new ComboBox<>(priorities);
27
28          Label lblSeverity = new Label("Severity");
29          ObservableList<String> severities =
FXCollections.observableArrayList("Blocker", "Workaround", "N/A");
30          ComboBox<String> cbSeverity = new ComboBox<>(severities);
31
32          Label lblCategory = new Label("Category");
33          ObservableList<String> categories = FXCollections.observableArrayList("Bug",
"Feature");
34          ComboBox<String> cbCategory = new ComboBox<>(categories);
35
36          Label lblProblem = new Label("Problem");
37          TextField tfProblem = new TextField();
38
39          Label lblDescription = new Label("Description");
40          TextArea taDescription = new TextArea();
41
42          gp.add( lblTitle,       1, 0);  // empty item at 0,0
43
44          gp.add( lblEmail,       0, 1);
45          gp.add(tfEmail,         1, 1);
46          gp.add( lblContract,    4, 1 );
47          gp.add( tfContract,     5, 1 );
48
49          gp.add( lblPriority,    0, 2);
50          gp.add( cbPriority,     1, 2);
51          gp.add( lblSeverity,    2, 2);
52          gp.add( cbSeverity,     3, 2);
53          gp.add( lblCategory,    4, 2);
54          gp.add( cbCategory,     5, 2);
55
56          gp.add( lblProblem,     0, 3); gp.add( tfProblem,     1, 3);
57          gp.add( lblDescription, 0, 4); gp.add( taDescription, 1, 4);
58
59          GridPane.setColumnSpan( lblTitle, 5 );
60          GridPane.setColumnSpan( tfEmail, 3 );
61          GridPane.setColumnSpan( tfProblem, 5 );
62          GridPane.setColumnSpan( taDescription, 5 );
63
64          Separator sep = new Separator(); // hr
65
```

```
66          ButtonBar buttonBar = new ButtonBar();
67          buttonBar.setPadding( new Insets(10) );
68
69          Button saveButton = new Button("Save");
70          Button cancelButton = new Button("Cancel");
71
72          buttonBar.setButtonData(saveButton, ButtonBar.ButtonData.OK_DONE);
73          buttonBar.setButtonData(cancelButton, ButtonBar.ButtonData.CANCEL_CLOSE);
74
75          buttonBar.getButtons().addAll(saveButton, cancelButton);
76
77          vbox.getChildren().addAll( gp, sep, buttonBar );
78
79          Scene scene = new Scene(vbox);
80
81          primaryStage.setTitle("Grid Pane App");
82          primaryStage.setScene(scene);
83          primaryStage.setWidth( 736 );
84          primaryStage.setHeight( 414  );
85          primaryStage.show();
86
87      }
88
89      public static void main(String[] args) {
90          launch(args);
91      }
92  }
```

4.7. Restricciones de fila y columna de GridPane

Los artículos anteriores sobre `GridPane` cómo crear un diseño de dos columnas con nombres de campo en el lado izquierdo y valores de campo en el lado derecho. Ese ejemplo se amplió para agregar más controles a una fila determinada y para usar espacios en el contenido del controlador de expansión. Este artículo presenta un par de clases JavaFX `ColumnConstraints` y `RowConstraints`. Estas clases dan especificaciones adicionales a una fila o columna. En este ejemplo, una fila que contiene un `TextArea` tendrá todo el espacio adicional cuando se cambie el tamaño de la ventana. Las dos columnas se establecerán en anchos iguales.

Esta captura de pantalla muestra un ejemplo modificado de artículos anteriores. El programa de demostración de este artículo tiene una sensación rotativa en la que los nombres de los campos se emparejan con los valores de campo verticalmente (sobre los valores) en lugar de horizontalmente. La expansión de filas y columnas se usa para alinear elementos que son más grandes que una sola celda.



Los rectángulos rojos y el texto no forman parte de la interfaz de usuario. Están identificando secciones de la pantalla que se abordarán más adelante con ColumnConstraints y RowConstaints.

Este código es la creación de la `Scene` raíz y los `GridPane` objetos.

```
VBox vbox = new VBox();

GridPane gp = new GridPane();
gp.setPadding( new Insets(10) );
gp.setHgap( 4 );
gp.setVgap( 10 );

VBox.setVgrow(gp, Priority.ALWAYS );
```

Este código crea los objetos de control de la interfaz de usuario que se usan en el artículo. Tenga en cuenta que Priority ahora se implementa como un `VBox` RadioButtons contenedor.

```
Label lblTitle = new Label("Support Ticket");

Label lblEmail = new Label("Email");
TextField tfEmail = new TextField();

Label lblContract = new Label("Contract");
TextField tfContract = new TextField();

Label lblPriority = new Label("Priority");
RadioButton rbMedium = new RadioButton("Medium");
RadioButton rbHigh = new RadioButton("High");
RadioButton rbLow = new RadioButton("Low");
VBox priorityVBox = new VBox();
priorityVBox.setSpacing( 2 );
GridPane.setVgrow(priorityVBox, Priority.SOMETIMES);
priorityVBox.getChildren().addAll( lblPriority, rbMedium, rbHigh, rbLow );

Label lblSeverity = new Label("Severity");
ObservableList<String> severities =
    FXCollections.observableArrayList("Blocker", "Workaround", "N/A");
ComboBox<String> cbSeverity = new ComboBox<>(severities);

Label lblCategory = new Label("Category");
ObservableList<String> categories =
    FXCollections.observableArrayList("Bug", "Feature");
ComboBox<String> cbCategory = new ComboBox<>(categories);

Label lblProblem = new Label("Problem");
TextField tfProblem = new TextField();

Label lblDescription = new Label("Description");
TextArea taDescription = new TextArea();
```

Los pares de control de etiqueta y valor de correo electrónico, contrato, problema y descripción se colocan en una sola columna. Deben tomar el ancho completo del de `GridPane` modo que cada uno tenga su columnSpan establecido en 2.

```
GridPane.setColumnSpan( tfEmail, 2 );
GridPane.setColumnSpan( tfContract, 2 );
GridPane.setColumnSpan( tfProblem, 2 );
GridPane.setColumnSpan( taDescription, 2 );
```

Los nuevos botones de opción de prioridad se combinan horizontalmente con cuatro controles de gravedad y categoría. Esta configuración de rowSpan le indica a JavaFX que coloque el VBox que contiene el RadioButton en una celda combinada que tiene cuatro filas de altura.

```
GridPane.setRowSpan( priorityVBox, 4 );
```

4.7.1. Restricciones de fila

En este punto, el código refleja la captura de pantalla de la interfaz de usuario que se presenta en Ejemplo de aplicación que usa filas y columnas . Para reasignar el espacio adicional en la base del formulario, use un objeto RowConstraints para establecer Priority.ALWAYS en la fila del `TextArea`. Esto dará como resultado el `TextArea` crecimiento para llenar el espacio disponible con algo utilizable.

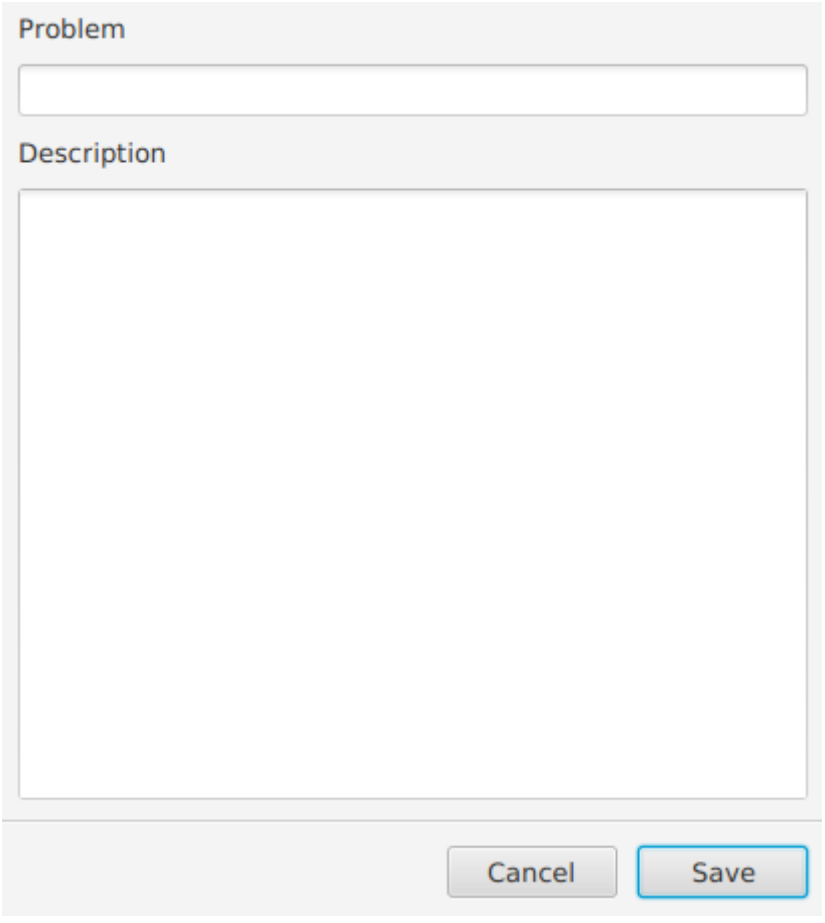


Figura 36. TextArea crece para llenar espacio adicional

Este código es un `RowConstraints` objeto para el `GridPane` para el `TextArea`. Antes del colocador, `RowConstraints` se asignan objetos para todas las demás filas. El método set de `getRowConstraints()` arrojará una excepción de índice cuando especifique la fila 12 sin asignar primero un objeto.

```
RowConstraints taDescriptionRowConstraints = new RowConstraints();
taDescriptionRowConstraints.setVgrow(Priority.ALWAYS);

for( int i=0; i<13; i++ ) {
    gp.getRowConstraints().add( new RowConstraints() );
}

gp.getRowConstraints().set( 12, taDescriptionRowConstraints );
```

Como sintaxis alternativa, hay un método setConstraints() disponible en `GridPane`. Esto pasará varios valores y obviará la necesidad de la llamada dedicada columnSpan set para `TextArea`. El `RowConstraints` código del listado anterior no aparecerá en el programa terminado.

```
gp.setConstraints(taDescription,
                  0, 12,
                  2, 1,
                  HPos.LEFT, VPos.TOP,
                  Priority.SOMETIMES, Priority.ALWAYS);
```

Este código identifica el `Node` at (0,12) que es el `TextArea`. El `TextArea` abarcará 2 columnas pero solo 1 fila. Los HPos y Vpos están configurados en la PARTE SUPERIOR IZQUIERDA. Finalmente, el `Priority` de hgrow es A VECES y el de vgrow es SIEMPRE. Dado que `TextArea` es la única fila con "SIEMPRE", obtendrá el espacio adicional. Si hubiera otras configuraciones SIEMPRE, el espacio se compartiría entre varias filas.

4.7.2. Restricciones de columna

Para asignar correctamente el espacio que rodea los controles de Severidad y Categoría, se especificarán ColumnConstraints. El comportamiento predeterminado asigna menos espacio a la primera columna debido a los botones de opción de prioridad más pequeños. La siguiente estructura alámbrica muestra el diseño deseado que tiene columnas iguales separadas por un margen de 4 píxeles (Hgap).

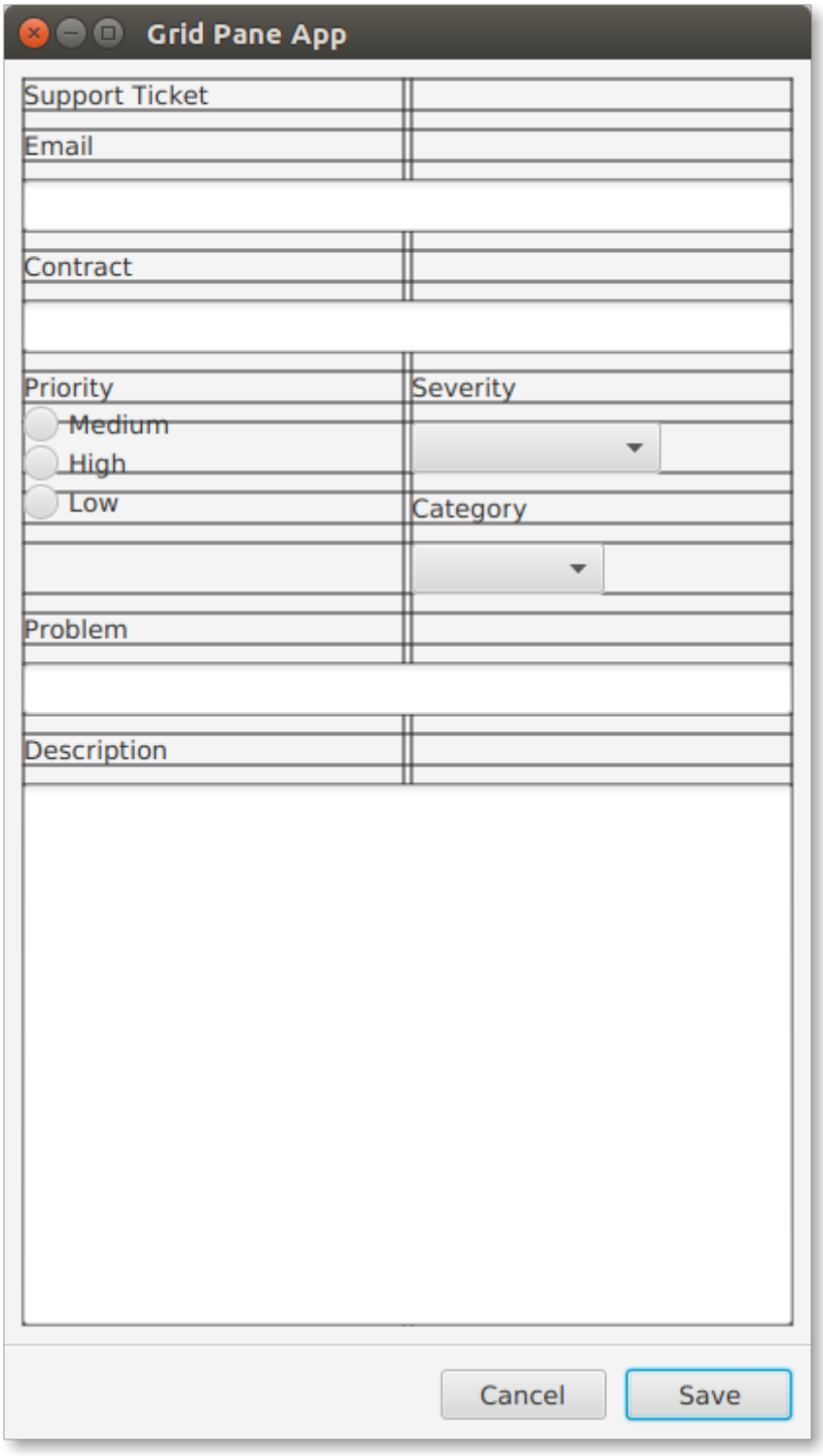
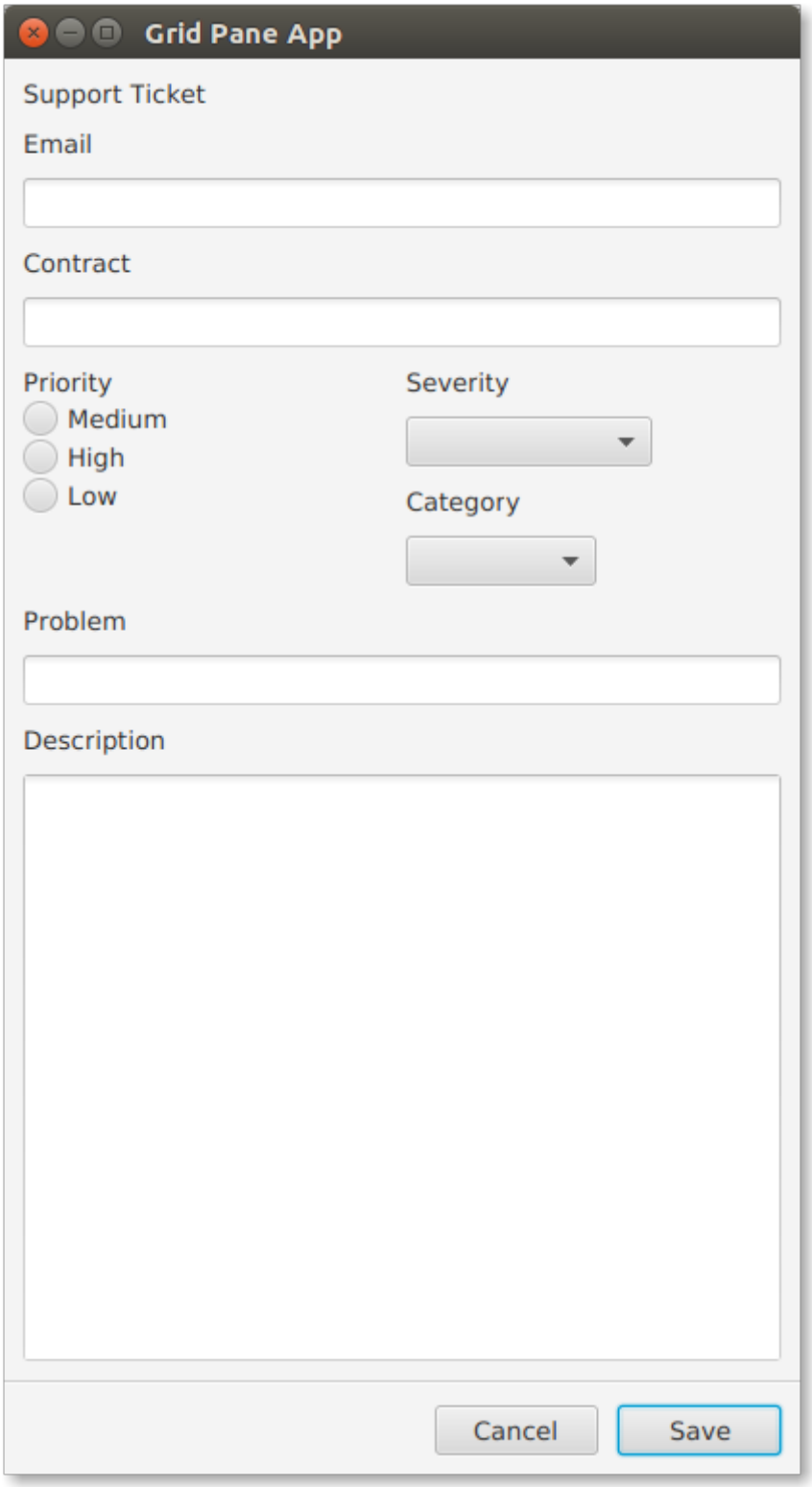


Figura 37. Wireframe de la aplicación de demostración

Para que los anchos de las columnas sean iguales, defina dos `ColumnConstraint` objetos y use un especificador de porcentaje.

```
ColumnConstraints col1 = new ColumnConstraints();
col1.setPercentWidth( 50 );
ColumnConstraints col2 = new ColumnConstraints();
col2.setPercentWidth( 50 );
gp.getColumnConstraints().addAll( col1, col2 );
```

Esta es una captura de pantalla del ejemplo terminado.



`GridPane` es un control importante en el desarrollo de aplicaciones empresariales JavaFX. Cuando trabaje en un requisito que involucre pares de nombre/valor y una sola vista de registro, use `GridPane`. Si bien `GridPane` es más fácil de usar que el `GridBagLayout` de Swing, todavía encuentro que la API es un poco inconveniente (asignación de índices propios, restricciones disociadas). Afortunadamente, existe Scene Builder que simplifica enormemente la construcción de este formulario.

4.7.3. Código completado

```
public class ConstraintsGridPaneApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        VBox vbox = new VBox();

        GridPane gp = new GridPane();
        gp.setPadding( new Insets(10) );
        gp.setHgap( 4 );
        gp.setVgap( 10 );

        VBox.setVgrow(gp, Priority.ALWAYS );

        Label lblTitle = new Label("Support Ticket");

        Label lblEmail = new Label("Email");
        TextField tfEmail = new TextField();

        Label lblContract = new Label("Contract");
        TextField tfContract = new TextField();

        Label lblPriority = new Label("Priority");
        RadioButton rbMedium = new RadioButton("Medium");
        RadioButton rbHigh = new RadioButton("High");
        RadioButton rbLow = new RadioButton("Low");
        VBox priorityVBox = new VBox();
        priorityVBox.setSpacing( 2 );
        GridPane.setVgrow(priorityVBox, Priority.SOMETIMES);
        priorityVBox.getChildren().addAll( lblPriority, rbMedium, rbHigh, rbLow );

        Label lblSeverity = new Label("Severity");
        ObservableList<String> severities =
FXCollections.observableArrayList("Blocker", "Workaround", "N/A");
        ComboBox<String> cbSeverity = new ComboBox<>(severities);

        Label lblCategory = new Label("Category");
        ObservableList<String> categories = FXCollections.observableArrayList("Bug",
"Feature");
        ComboBox<String> cbCategory = new ComboBox<>(categories);

        Label lblProblem = new Label("Problem");
        TextField tfProblem = new TextField();

        Label lblDescription = new Label("Description");
        TextArea taDescription = new TextArea();

        gp.add( lblTitle,       0, 0);

```

```
48          gp.add( lblEmail,       0, 1);
49          gp.add(tfEmail,         0, 2);
50
51          gp.add( lblContract,    0, 3 );
52          gp.add( tfContract,     0, 4 );
53
54          gp.add( priorityVBox,   0, 5);
55
56          gp.add( lblSeverity,    1, 5);
57          gp.add( cbSeverity,     1, 6);
58          gp.add( lblCategory,    1, 7);
59          gp.add( cbCategory,     1, 8);
60
61          gp.add( lblProblem,     0, 9);
62          gp.add( tfProblem,      0, 10);
63
64          gp.add( lblDescription, 0, 11);
65          gp.add( taDescription,  0, 12);
66
67          GridPane.setColumnSpan( tfEmail, 2 );
68          GridPane.setColumnSpan( tfContract, 2 );
69          GridPane.setColumnSpan( tfProblem, 2 );
70
71          GridPane.setRowSpan( priorityVBox, 4 );
72
73          gp.setConstraints(taDescription,
74                            0, 12,
75                            2, 1,
76                            HPos.LEFT, VPos.TOP,
77                            Priority.SOMETIMES, Priority.ALWAYS);
78
79          ColumnConstraints col1 = new ColumnConstraints();
80          col1.setPercentWidth( 50 );
81          ColumnConstraints col2 = new ColumnConstraints();
82          col2.setPercentWidth( 50 );
83          gp.getColumnConstraints().addAll( col1, col2 );
84
85          Separator sep = new Separator(); // hr
86
87          ButtonBar buttonBar = new ButtonBar();
88          buttonBar.setPadding( new Insets(10) );
89
90          Button saveButton = new Button("Save");
91          Button cancelButton = new Button("Cancel");
92
93          buttonBar.setButtonData(saveButton, ButtonBar.ButtonData.OK_DONE);
94          buttonBar.setButtonData(cancelButton, ButtonBar.ButtonData.CANCEL_CLOSE);
95
96          buttonBar.getButtons().addAll(saveButton, cancelButton);
97
98          vbox.getChildren().addAll( gp, sep, buttonBar );
99
```

```
100         Scene scene = new Scene(vbox);
101 
102         primaryStage.setTitle("Grid Pane App");
103         primaryStage.setScene(scene);
104         primaryStage.setWidth( 414 );
105         primaryStage.setHeight( 736  );
106         primaryStage.show();
107 
108     }
109 
110     public static void main(String[] args) {
111         launch(args);
112     }
113 }
```

4.8. AnchorPane

`AnchorPane` es un control contenedor que define su diseño en términos de bordes. Cuando se coloca en un contenedor, `AnchorPane` se estira para llenar el espacio disponible. Los hijos de `AnchorPane` expresan sus posiciones y tamaños como distancias desde los bordes: Arriba, Izquierda, Abajo, Derecha. Si se colocan una o dos configuraciones de anclaje en un `AnchorPane` niño, el niño se fijará a esa esquina de la ventana. Si se utilizan más de dos configuraciones de anclaje, el niño se estirará para llenar el espacio horizontal y vertical disponible.

Esta maqueta muestra un `TextArea` rodeado por un conjunto de controles: un `Hyperlink` y dos indicadores de estado. Dado `TextArea` que contendrá todo el contenido, debería ocupar la mayor parte del espacio inicialmente y debería adquirir cualquier espacio adicional de un cambio de tamaño. En la periferia, hay una `Hyperlink` en la parte superior derecha, una conexión `Label` y `Circle` en la parte inferior derecha y un estado `Label` en la parte inferior izquierda.

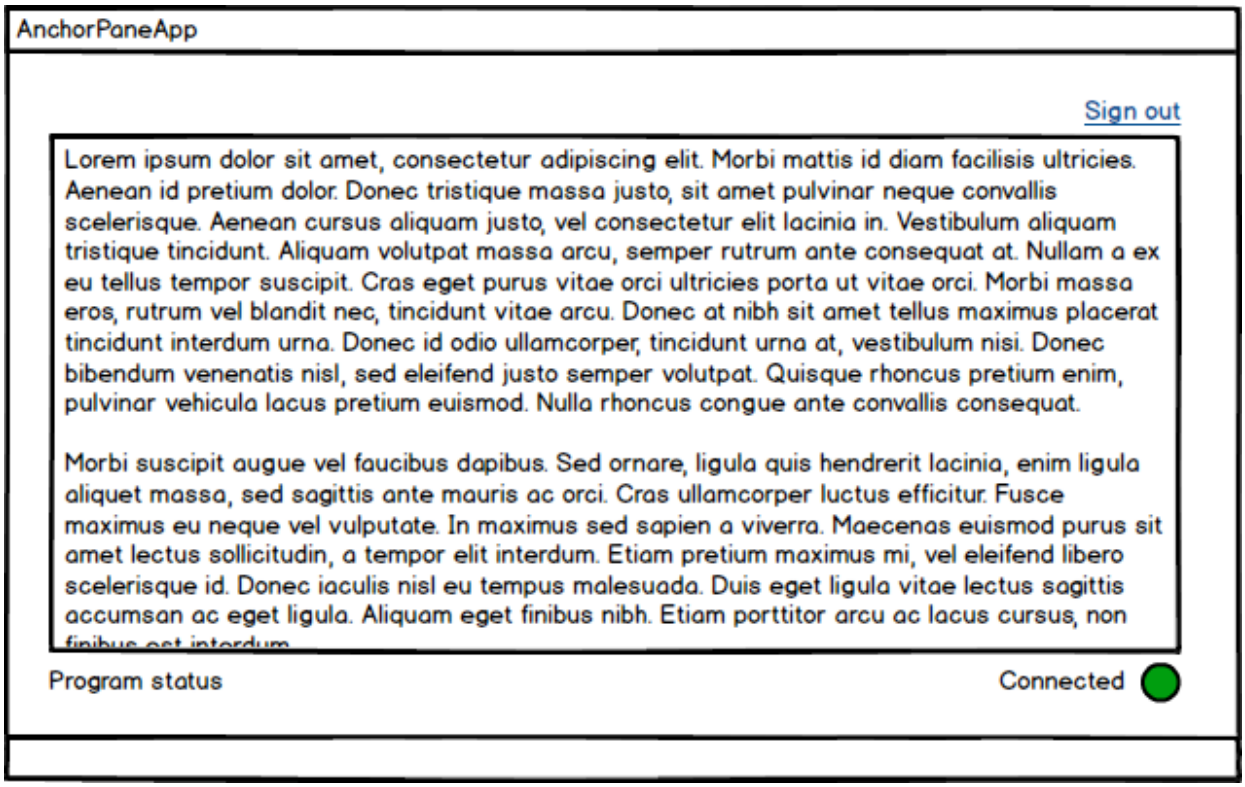


Figura 39. AnchorPane con TextArea

4.8.1. anclas

Para comenzar el diseño, cree un `AnchorPane` objeto y agréguelo al archivo `Scene`.

```
AnchorPane ap = new AnchorPane();
Scene scene = new Scene(ap);
```

Los anclajes se establecen mediante métodos estáticos de la clase AnchorPane. Los métodos, uno por borde, aceptan el `Node` y un desplazamiento. Para el `Hyperlink`, se establecerá un ancla en el borde superior y otra en el borde derecho. Se establece un desplazamiento de 10,0 para cada borde para que el enlace no se comprima contra el lado.

```
Hyperlink signoutLink = new Hyperlink("Sign Out");

ap.getChildren().add( signoutLink );

AnchorPane.setTopAnchor( signoutLink, 10.0d );
AnchorPane.setRightAnchor( signoutLink, 10.0d );
```

Cuando se cambia el tamaño de la pantalla, AnchorPane cambiará de tamaño y signoutLink mantendrá su posición superior derecha. Debido a que no se especifican los anclajes izquierdo ni inferior, signoutLink no se estirará.

A continuación, se añaden la conexión `Label` y `Circle`. Estos controles están envueltos en un archivo `HBox`.

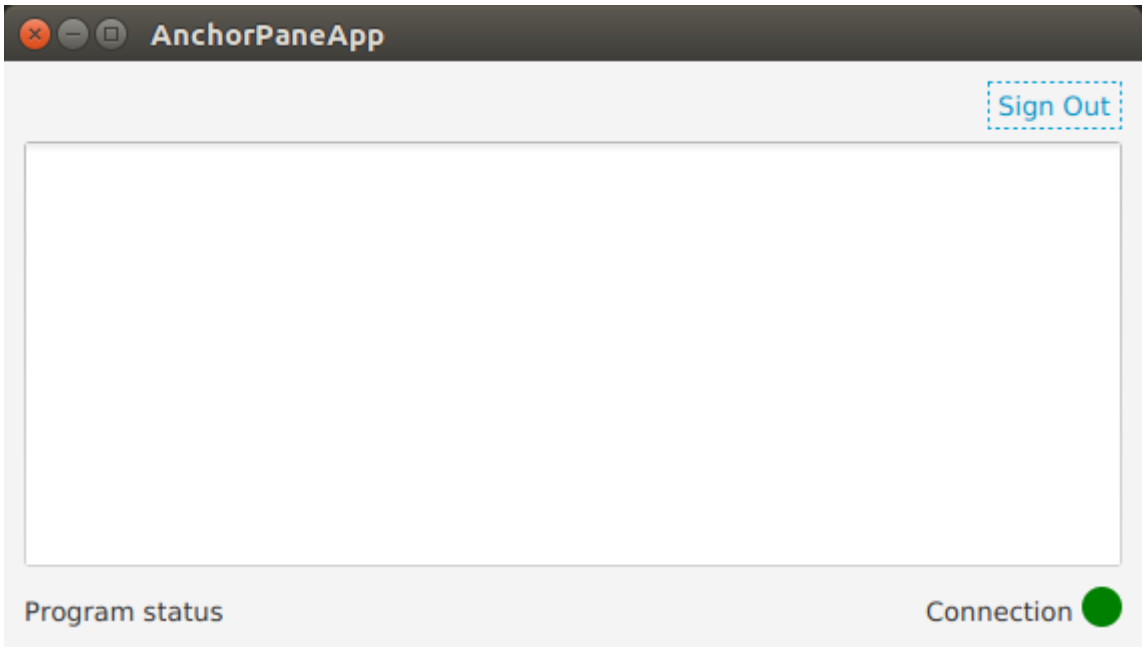
```
Circle circle = new Circle();
circle.setFill(Color.GREEN );
circle.setRadius(10);

Label connLabel = new Label("Connection");

HBox connHBox = new HBox();
connHBox.setSpacing( 4.0d );
connHBox.setAlignment(Pos.BOTTOM_RIGHT);
connHBox.getChildren().addAll( connLabel, circle );

AnchorPane.setBottomAnchor( connHBox, 10.0d );
AnchorPane.setRightAnchor( connHBox, 10.0d );

ap.getChildren().add( connHBox );
```

Al igual que con signoutLink, connHBox se fija en un lugar de la pantalla. connHBox se establece en 10 píxeles desde el borde inferior y 10 píxeles desde el borde derecho.

Se agrega el estado inferior izquierdo `Label`. Los anclajes izquierdo e inferior están establecidos.

```
Label statusLabel = new Label("Program status");
ap.getChildren().add( statusLabel );

AnchorPane.setBottomAnchor( statusLabel, 10.0d );
AnchorPane.setLeftAnchor( statusLabel, 10.0d );
```

Esta es una captura de pantalla de la aplicación terminada. Las etiquetas de estado y control se encuentran en la parte inferior de la pantalla, fijadas a los bordes izquierdo y derecho respectivamente. está anclado en la `Hyperlink`parte superior derecha.



4.8.2. Cambiar el tamaño

Los controles en la periferia pueden variar en tamaño. Por ejemplo, un mensaje de estado o un mensaje de conexión puede ser más largo. Sin embargo, la longitud adicional se puede acomodar en este diseño extendiendo el estado de la parte inferior izquierda `Label` hacia la derecha y extendiendo el estado de conexión de la parte inferior derecha `Label` hacia la izquierda. Cambiar el tamaño con este diseño moverá estos controles en términos absolutos, pero se adherirán a sus respectivos bordes más el desplazamiento.

Ese no es el caso con el `TextArea`. Debido a que `TextArea` puede contener una gran cantidad de contenido, debe recibir cualquier espacio adicional que el usuario le dé a la ventana. Este control estará anclado a las cuatro esquinas del `AnchorPane`. Esto hará `TextArea` que cambie el tamaño cuando la ventana cambie de tamaño. se fija en la `TextArea` parte superior izquierda y, a medida que el usuario arrastra los controladores de la ventana hacia la parte inferior derecha, la esquina inferior derecha de los `TextArea` movimientos también.

Esta imagen muestra el resultado de dos operaciones de cambio de tamaño. La captura de pantalla superior es un cambio de tamaño vertical al arrastrar el borde inferior de la ventana hacia abajo. La captura de pantalla inferior es un cambio de tamaño horizontal al arrastrar el borde derecho de la ventana hacia la derecha.

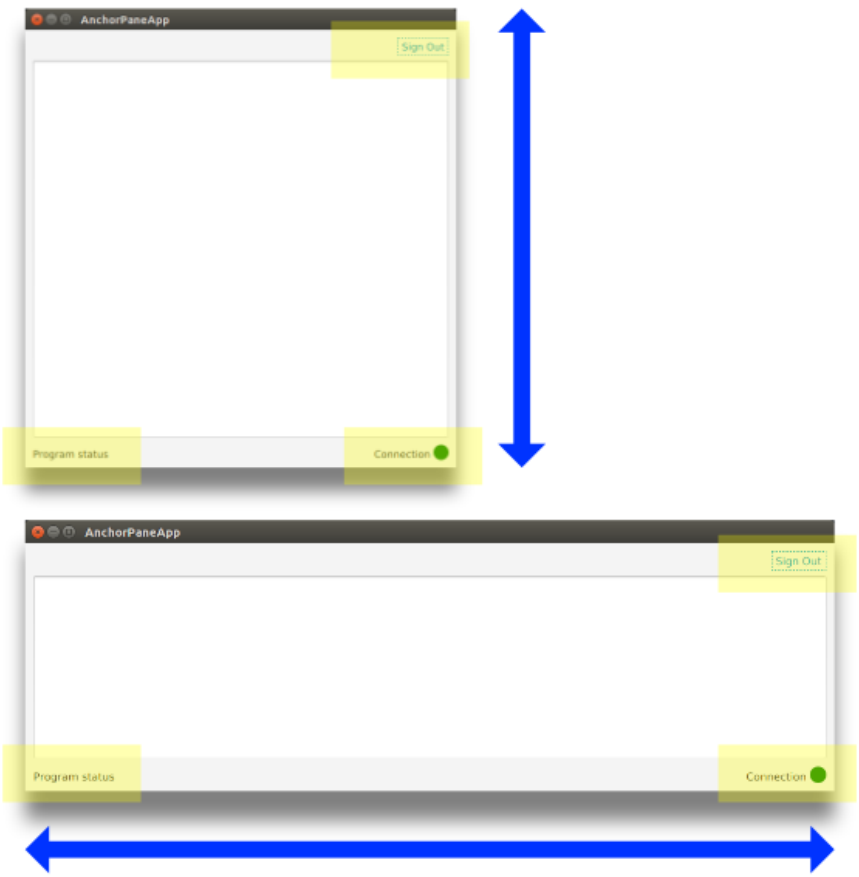


Figura 41. "Aplicación AnchorPane redimensionada

Los cuadros resaltados muestran que los controles que bordean `TextArea` conservan sus posiciones relativas a los bordes. El `TextArea` mismo se redimensiona en función del redimensionamiento de la ventana. Las compensaciones superior e inferior de la `TextArea` cuenta para los otros controles para que no se oculten.

```
TextArea ta = new TextArea();

AnchorPane.setTopAnchor( ta, 40.0d );
AnchorPane.setBottomAnchor( ta, 40.0d );
AnchorPane.setRightAnchor( ta, 10.0d );
AnchorPane.setLeftAnchor( ta, 10.0d );

ap.getChildren().add( ta );
```

`AnchorPane` es una buena opción cuando tiene una mezcla de niños de tamaño variable y de posición fija. Se prefieren otros controles como `VBox` y `HBox` con una `Priority` configuración si solo hay un niño que necesita cambiar el tamaño. Utilice estos controles en lugar de `AnchorPane` con un solo niño que tenga las cuatro anclas configuradas. Recuerda que para establecer un ancla en un niño, usas un método estático de la clase contenedora como AnchorPane.setTopAnchor().

4.8.3. Código completado

El siguiente es el código completo para el `AnchorPane` ejemplo.

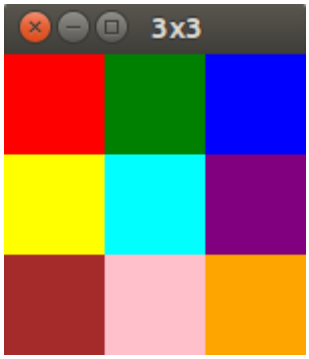
```
public class AnchorPaneApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        AnchorPane ap = new AnchorPane();

        // upper-right sign out control
        Hyperlink signoutLink = new Hyperlink("Sign Out");

        ap.getChildren().add( signoutLink );

        AnchorPane.setTopAnchor( signoutLink, 10.0d );
        AnchorPane.setRightAnchor( signoutLink, 10.0d );

        // lower-left status label
        Label statusLabel = new Label("Program status");
        ap.getChildren().add( statusLabel );

        AnchorPane.setBottomAnchor( statusLabel, 10.0d );
        AnchorPane.setLeftAnchor( statusLabel, 10.0d );

        // lower-right connection status control
        Circle circle = new Circle();
        circle.setFill(Color.GREEN );
        circle.setRadius(10);
```

```
27
28          Label connLabel = new Label("Connection");
29
30          HBox connHBox = new HBox();
31          connHBox.setSpacing( 4.0d );
32          connHBox.setAlignment(Pos.BOTTOM_RIGHT);
33          connHBox.getChildren().addAll( connLabel, circle );
34
35          AnchorPane.setBottomAnchor( connHBox, 10.0d );
36          AnchorPane.setRightAnchor( connHBox, 10.0d );
37
38          ap.getChildren().add( connHBox );
39
40          // top-left content; takes up extra space
41          TextArea ta = new TextArea();
42          ap.getChildren().add( ta );
43
44          AnchorPane.setTopAnchor( ta, 40.0d );
45          AnchorPane.setBottomAnchor( ta, 40.0d );
46          AnchorPane.setRightAnchor( ta, 10.0d );
47          AnchorPane.setLeftAnchor( ta, 10.0d );
48
49          Scene scene = new Scene(ap);
50
51          primaryStage.setTitle("AnchorPaneApp");
52          primaryStage.setScene( scene );
53          primaryStage.setWidth(568);
54          primaryStage.setHeight(320);
55          primaryStage.show();
56      }
57
58      public static void main(String[] args) {
59          launch(args);
60      }
61  }
```

4.9. TilePane (panel de mosaico)

A `TilePane` se utiliza para el diseño de cuadrícula de celdas de igual tamaño. Las propiedades prefColumns y prefRows definen el número de filas y columnas en la cuadrícula. Para agregar nodos a `TilePane`, acceda a la propiedad child y llame al método add() o addAll(). Esto es más fácil de usar que `GridPane` lo que requiere una configuración explícita de la posición de fila/columna de los nodos.

Esta captura de pantalla muestra una `TilePane` cuadrícula definida como de tres por tres. El `TilePane` contiene nueve `Rectangle` objetos.



A continuación se muestra el código completo para la cuadrícula de tres por tres. La propiedad children de `TilePane` proporciona el método addAll() al que `Rectangle` se agregan los objetos. La propiedad tileAlignment coloca cada uno de los `Rectangle` objetos en el centro de su mosaico correspondiente.

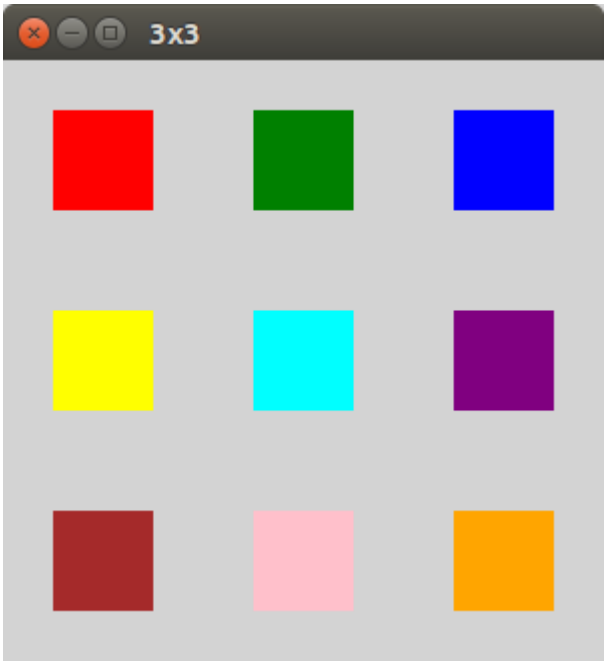
ThreeByThreeApp.java

```
1  public class ThreeByThreeApp extends Application {
2
3      @Override
4      public void start(Stage primaryStage) throws Exception {
5
6          TilePane tilePane = new TilePane();
7          tilePane.setPrefColumns(3);
8          tilePane.setPrefRows(3);
9          tilePane.setTileAlignment( Pos.CENTER );
10
11         tilePane.getChildren().addAll(
12                 new Rectangle(50, 50, Color.RED),
13                 new Rectangle( 50, 50, Color.GREEN ),
14                 new Rectangle( 50, 50, Color.BLUE ),
15                 new Rectangle( 50, 50, Color.YELLOW ),
16                 new Rectangle( 50, 50, Color.CYAN ),
17                 new Rectangle( 50, 50, Color.PURPLE ),
18                 new Rectangle( 50, 50, Color.BROWN ),
19                 new Rectangle( 50, 50, Color.PINK ),
20                 new Rectangle( 50, 50, Color.ORANGE )
21         );
22
23         Scene scene = new Scene(tilePane);
24         scene.setFill(Color.LIGHTGRAY);
25
26         primaryStage.setTitle("3x3");
27         primaryStage.setScene( scene );
28         primaryStage.show();
29     }
30
31     public static void main(String[] args) {launch(args);}
32 }
```

Dado que todo el `Node` contenido de los `TilePane` Rectángulos era del mismo tamaño, el diseño está empaquetado y la configuración de TileAlignment no se nota. Cuando las propiedades tilePrefHeight y tilePrefWidth se configuran para que sean más grandes que el contenido, digamos mosaicos de 100x100 que contienen rectángulos de 50x50, tileAlignment determinará cómo se usará el espacio adicional.

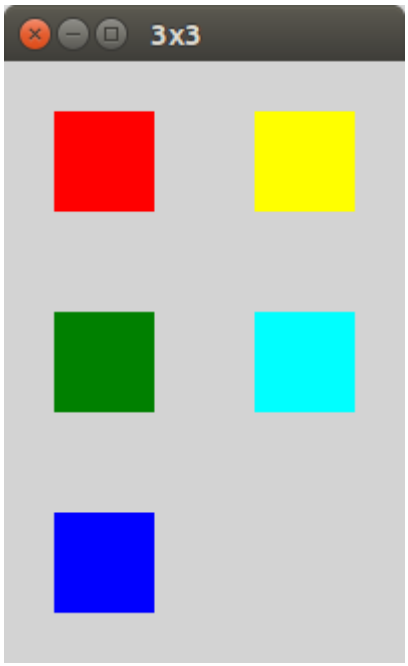
Consulte la siguiente clase ThreeByThreeApp modificada que establece el tilePrefHeight y el tilePrefWidth.

```
1         tilePane.setPrefTileHeight(100);
2         tilePane.setPrefTileWidth(100);
```



En las capturas de pantalla anteriores, se proporcionaron nueve objetos Rectangle a la cuadrícula de tres por tres. Si el contenido no coincide con la `TilePane` definición, esas celdas colapsarán. Esta modificación agrega solo cinco Rectángulos en lugar de nueve. La primera fila contiene contenido para los tres mosaicos. La segunda fila tiene contenido solo para los dos primeros archivos. Falta la tercera fila por completo.

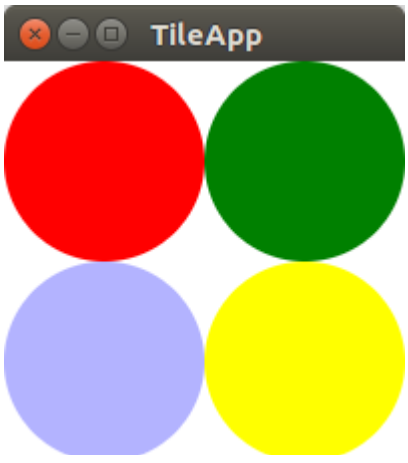
Hay una propiedad de "orientación" que indica `TilePane` agregar elementos fila por fila (HORIZONTAL, el valor predeterminado) o columna por columna (VERTICAL). Si se usa VERTICAL, la primera columna tendrá tres elementos, la segunda columna tendrá solo los dos superiores y faltará la tercera columna. Esta captura de pantalla muestra los cinco rectángulos que se agregan a la cuadrícula de tres por tres (nueve mosaicos) con orientación VERTICAL.



4.9.1. Algoritmos

Es posible crear diseños de cuadrícula JavaFX con otros contenedores como `GridPane`, `VBox` y `HBox`. TilePane es una conveniencia que define el diseño de la cuadrícula de antemano y hace que agregar elementos a la cuadrícula sea una simple llamada add() o addAll(). A diferencia de un diseño de cuadrícula creado con una combinación de anidados `VBox` y `HBox` contenedores, los `TilePane` contenidos son elementos secundarios directos. Esto facilita el bucle sobre los niños durante el procesamiento de eventos, lo que ayuda a implementar ciertos algoritmos.

Esta aplicación de ejemplo coloca cuatro círculos en un archivo `TilePane`. Se adjunta un controlador de eventos `TilePane` que busca una selección de uno de los círculos. Si se selecciona un Círculo, se atenúa a través de la configuración de opacidad. Si se vuelve a seleccionar el Círculo, se restaura su color original. Esta captura de pantalla muestra la aplicación con el azul `Circle` que aparece de color púrpura porque se ha seleccionado.



El programa comienza agregando los elementos y configurando una propiedad personalizada "seleccionada" utilizando la API de flujo de Java 8.

TileApp.java

```
        TilePane tilePane = new TilePane();
        tilePane.setPrefColumns(2);
        tilePane.setPrefRows(2);
        tilePane.setTileAlignment( Pos.CENTER );

        Circle redCircle = new Circle(50, Color.RED);
        Circle greenCircle = new Circle( 50, Color.GREEN );
        Circle blueCircle = new Circle( 50, Color.BLUE );
        Circle yellowCircle = new Circle( 50, Color.YELLOW );

        List<Circle> circles = new ArrayList<>();
        circles.add( redCircle );
        circles.add( greenCircle );
        circles.add( blueCircle );
        circles.add( yellowCircle );

        circles
                .stream()
                .forEach( (c) -> c.getProperties().put( "selected", Boolean.FALSE ));

        tilePane.getChildren().addAll(
               circles
        );
```

A continuación, el controlador de eventos se adjunta al evento del mouse. Esto también está usando Java 8 Streams. El método filter() determina si `Circle` se selecciona o no usando el método Node.contains() en las coordenadas convertidas. Si esa expresión pasa, se usa findFirst() para recuperar la primera (y en este caso, la única) coincidencia. El bloque de código en ifPresent() establece el indicador "seleccionado" para realizar un seguimiento del `Circle` estado y ajusta la opacidad.

TileApp.java

```
    tilePane.setOnMouseClicked(

        (evt) -> tilePane
                    .getChildren()
                    .stream()
                    .filter( c ->
                        c.contains(
                          c.sceneToLocal(evt.getSceneX(), evt.getSceneY(), true)
                        )
                     )
                    .findFirst()
                    .ifPresent(
                            (c) -> {
```

```
                                Boolean selected = (Boolean)
c.getProperties().get("selected");
                                if( selected == null || selected == Boolean.FALSE ) {
                                    c.setOpacity(0.3d);
                                    c.getProperties().put("selected", Boolean.TRUE);
                                } else {
                                    c.setOpacity( 1.0d );
                                    c.getProperties().put("selected", Boolean.FALSE);
                                }
                            }
                    )
    );
```

4.9.2. otro controlador

Dado que el programa guarda los círculos en colecciones de Java `List`, el `TilePane` contenido se puede reemplazar con llamadas allAll() repetidas. Este controlador de eventos se activa cuando el usuario presiona una "S" en el archivo `Scene`. El contenido del respaldo `List` se mezcla y se vuelve a agregar al archivo `TilePane`.

TileApp.java

```
        scene.setOnKeyPressed(
                (evt) -> {
                    if( evt.getCode().equals(KeyCode.S) ) {
                        Collections.shuffle( circles );
                        tilePane.getChildren().clear();
                        tilePane.getChildren().addAll( circles );
                    }
                }
        );
```

Si bien es factible, una cuadrícula construida con VBoxes y HBoxes sería un poco más difícil debido a las estructuras anidadas. Además, `TilePane` no estirará el contenido para llenar espacio adicional, lo que lo hace adecuado para controles compuestos que deben empaquetarse juntos por razones ergonómicas.

`TilePane` crea un diseño basado en cuadrícula de celdas de igual tamaño. Los contenidos se agregan en `TilePane` función de la configuración de prefRows, prefColumns y orientación. Si la cuadrícula contiene más mosaicos que nodos agregados, habrá espacios en el diseño y las filas y columnas pueden contraerse si no se proporcionó contenido alguno. Esta publicación mostró un par de algoritmos que se implementaron fácilmente debido a la interfaz simple de TilePane.

4.9.3. Código completo

A continuación se muestra el código completo de TileApp.

TileApp.java (completa)

```
public class TileApp extends Application {
```

```
    @Override
    public void start(Stage primaryStage) throws Exception {

        TilePane tilePane = new TilePane();
        tilePane.setPrefColumns(2);
        tilePane.setPrefRows(2);
        tilePane.setTileAlignment( Pos.CENTER );

        Circle redCircle = new Circle(50, Color.RED);
        Circle greenCircle = new Circle( 50, Color.GREEN );
        Circle blueCircle = new Circle( 50, Color.BLUE );
        Circle yellowCircle = new Circle( 50, Color.YELLOW );

        List<Circle> circles = new ArrayList<>();
        circles.add( redCircle );
        circles.add( greenCircle );
        circles.add( blueCircle );
        circles.add( yellowCircle );

        circles
                .stream()
                .forEach( (c) -> c.getProperties().put( "selected", Boolean.FALSE ));

        tilePane.getChildren().addAll(
               circles
        );

        tilePane.setOnMouseClicked(

            (evt) -> tilePane
                        .getChildren()
                        .stream()
                        .filter( c ->
                            c.contains(
                              c.sceneToLocal(evt.getSceneX(), evt.getSceneY(), true)
                            )
                         )
                        .findFirst()
                        .ifPresent(
                                (c) -> {
                                    Boolean selected = (Boolean)
c.getProperties().get("selected");
                                    if( selected == null || selected == Boolean.FALSE )
{
                                        c.setOpacity(0.3d);
                                        c.getProperties().put("selected",
Boolean.TRUE);
                                    } else {
                                        c.setOpacity( 1.0d );
                                        c.getProperties().put("selected",
Boolean.FALSE);
```

```
                                    }
                                }
                        )
        );

        Scene scene = new Scene(tilePane);

        scene.setOnKeyPressed(
                (evt) -> {
                    if( evt.getCode().equals(KeyCode.S) ) {
                        Collections.shuffle( circles );
                        tilePane.getChildren().clear();
                        tilePane.getChildren().addAll( circles );
                    }
                }
        );

        primaryStage.setTitle("TileApp");
        primaryStage.setScene( scene );
        primaryStage.show();

    }

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

4.10. TitledPane

A `TitledPane` es un `Node` contenedor emparejado con a `Label` y un control opcional para mostrar y ocultar el contenido del contenedor. Dado que `TitledPane` está limitado a un solo `Node`, a menudo se combina con un contenedor que admite varios elementos secundarios como `VBox`. Funcionalmente, puede ocultar detalles no esenciales de un formulario o controles relacionados con grupos.

Este ejemplo es una aplicación de búsqueda web que acepta un conjunto de palabras clave en un archivo `TextField`. El usuario presiona el botón Buscar para ejecutar una búsqueda. El Avanzado `TitlePane` se expande para proporcionar argumentos de búsqueda adicionales.

Esta captura de pantalla muestra el estado no expandido que es la vista para un usuario que ejecuta una búsqueda simple de palabras clave.

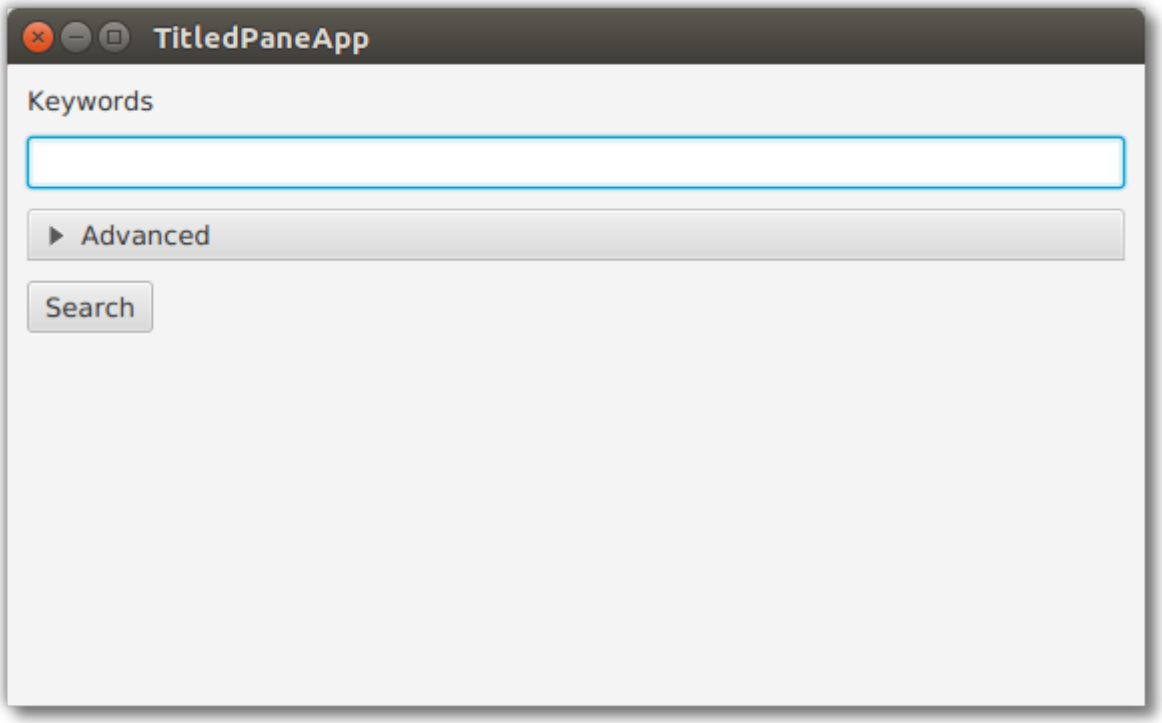
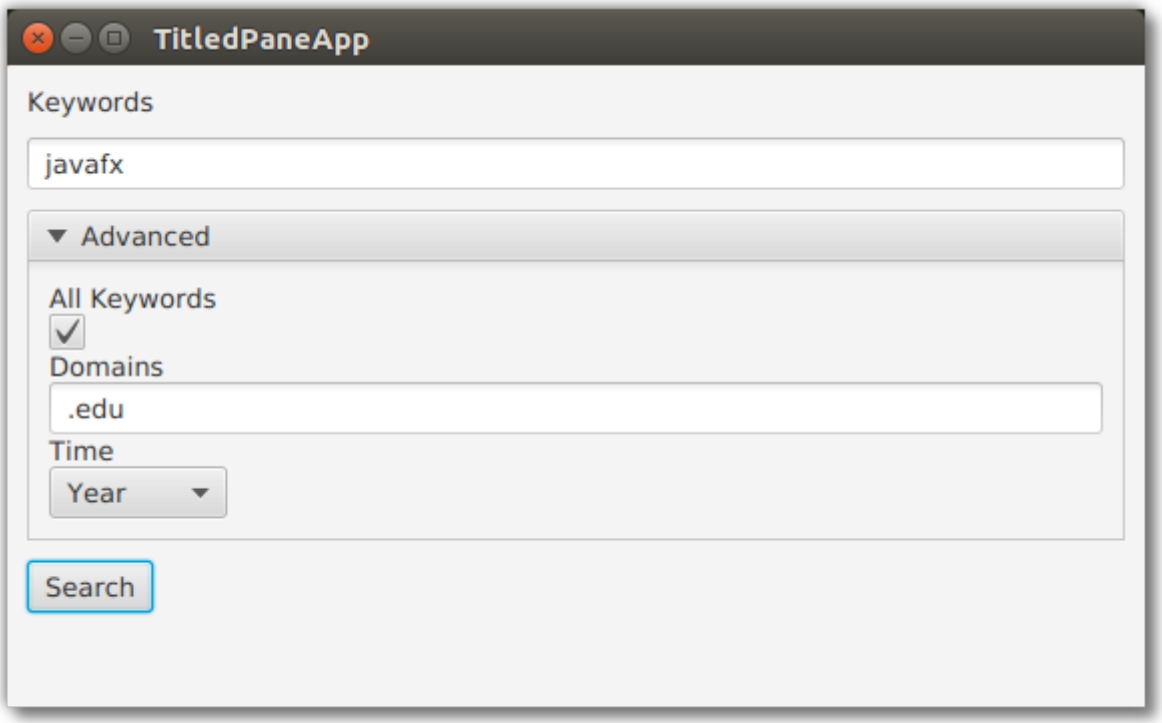


Figura 47. TitledPane sin expandir

La siguiente captura de pantalla muestra la vista para un usuario que requiere parámetros de búsqueda avanzada. El Advanced TitledPane se expandió presionando la flecha en el `TitledPane` encabezado.



Para crear un `TitledPane`, use el constructor para pasar un título de cadena y un solo `Node` hijo. También se puede usar el constructor predeterminado y el título y `Node` establecer usando setters. Este código usa el constructor parametrizado. A `VBox` es el único hijo de `TitledPane`. Sin embargo, el `VBox` mismo contiene varios controles.

TitledPaneApp.java

```
        VBox advancedVBox = new VBox(
                new Label("All Keywords"),
                new CheckBox(),
                new Label("Domains"),
                new TextField(),
                new Label("Time"),
                new ComboBox<>(
                    FXCollections.observableArrayList( "Day", "Month", "Year" )
                )
        );

        TitledPane titledPane = new TitledPane(
                "Advanced",
                advancedVBox
        );
        titledPane.setExpanded( false );
```

De forma predeterminada, `TitledPane` se expandirá. Esto no se ajusta al caso de uso de ocultar información no esencial, por lo que la propiedad expandida se establece después de que se crea el objeto.

4.10.1. Plegable

Otra propiedad de `TitledPane` es plegable. De forma predeterminada, la `TitledPane` propiedad contraíble se establece en verdadero. Sin embargo, se puede proporcionar una agrupación rápida a los controles que no son plegables. La siguiente captura de pantalla muestra este caso de uso.

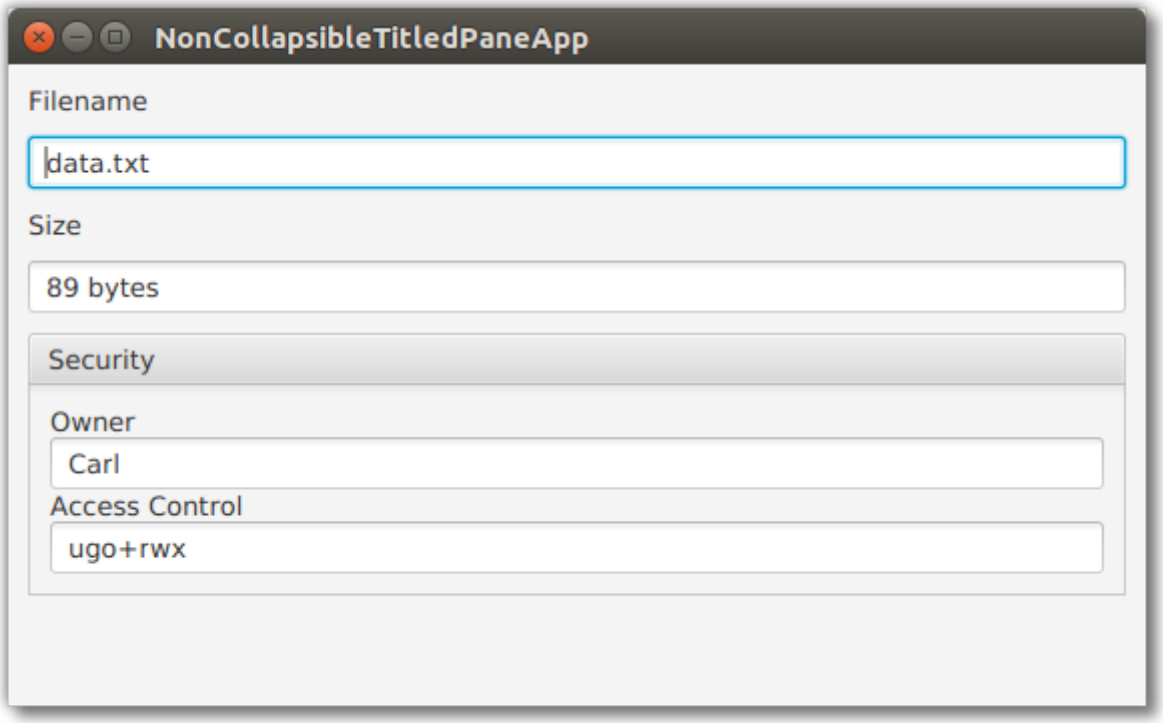


Figura 49. Conjunto contraíble en falso

Este código establece la bandera contraíble después de llamar al constructor.

```
            VBox securityVBox = new VBox(
                    new Label("Owner"),
                    new TextField(),
                    new Label("Access Control"),
                    new TextField()
            );

            TitledPane tp = new TitledPane("Security", securityVBox);
            tp.setCollapsible( false );
```

4.10.2. Código completo

El siguiente es el código completo para la primera demostración que involucra los parámetros de búsqueda ocultos "TitledPaneApp".

```
public class TitledPaneApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        VBox vbox = new VBox(
                new Label("Keywords" ),
                new TextField()
        );

```

```
11          vbox.setPadding( new Insets(10) );
12          vbox.setSpacing( 10 );
13
14          VBox advancedVBox = new VBox(
15                  new Label("All Keywords"),
16                  new CheckBox(),
17                  new Label("Domains"),
18                  new TextField(),
19                  new Label("Time"),
20                  new ComboBox<>(
21                      FXCollections.observableArrayList( "Day", "Month", "Year" )
22                  )
23          );
24
25          TitledPane titledPane = new TitledPane(
26                  "Advanced",
27                  advancedVBox
28          );
29          titledPane.setExpanded( false );
30
31          vbox.getChildren().addAll(
32                  titledPane,
33                  new Button("Search")
34          );
35
36          Scene scene = new Scene( vbox );
37
38          primaryStage.setTitle( "TitledPaneApp" );
39          primaryStage.setScene( scene );
40          primaryStage.setWidth( 568 );
41          primaryStage.setHeight( 320 );
42          primaryStage.show();
43      }
44
45      public static void main(String[] args) {
46          launch(args);
47      }
48  }
```

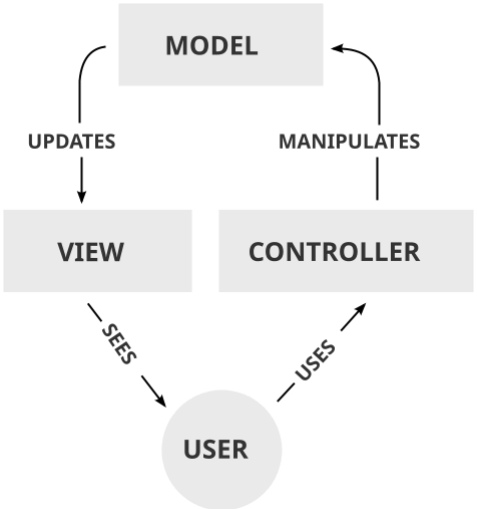
5. Estructura de la aplicación

5.1. El patrón MVVM

Modelo-vista-controlador (**MVC**) es un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (véase ***Middleware***).
- La **Vista**: Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.



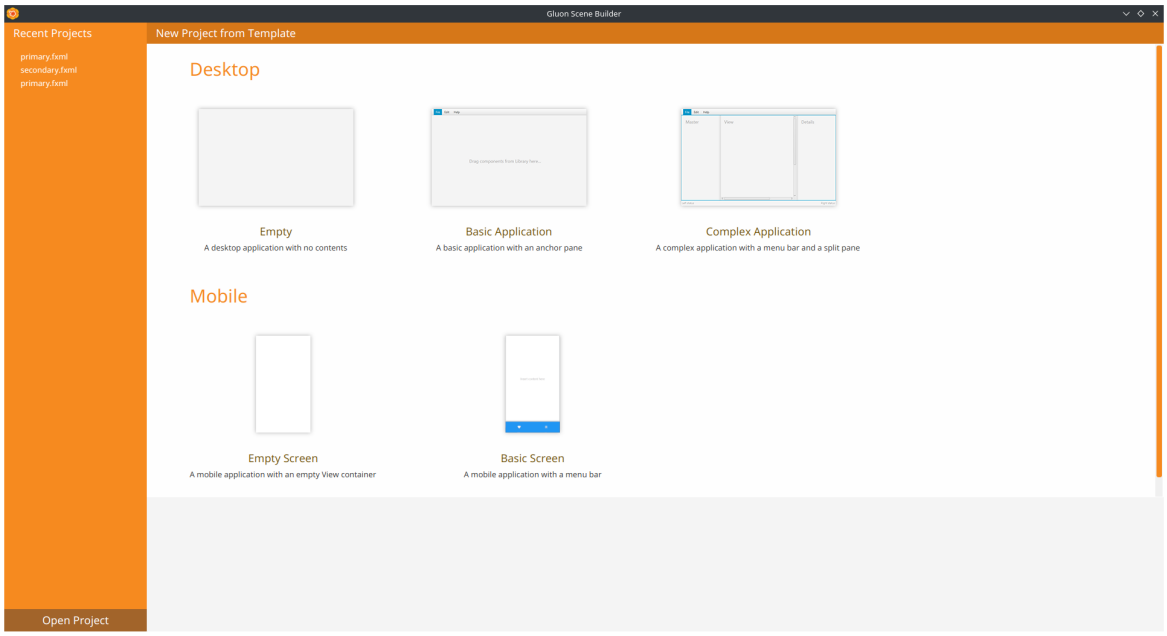
5.2. Scene Builder

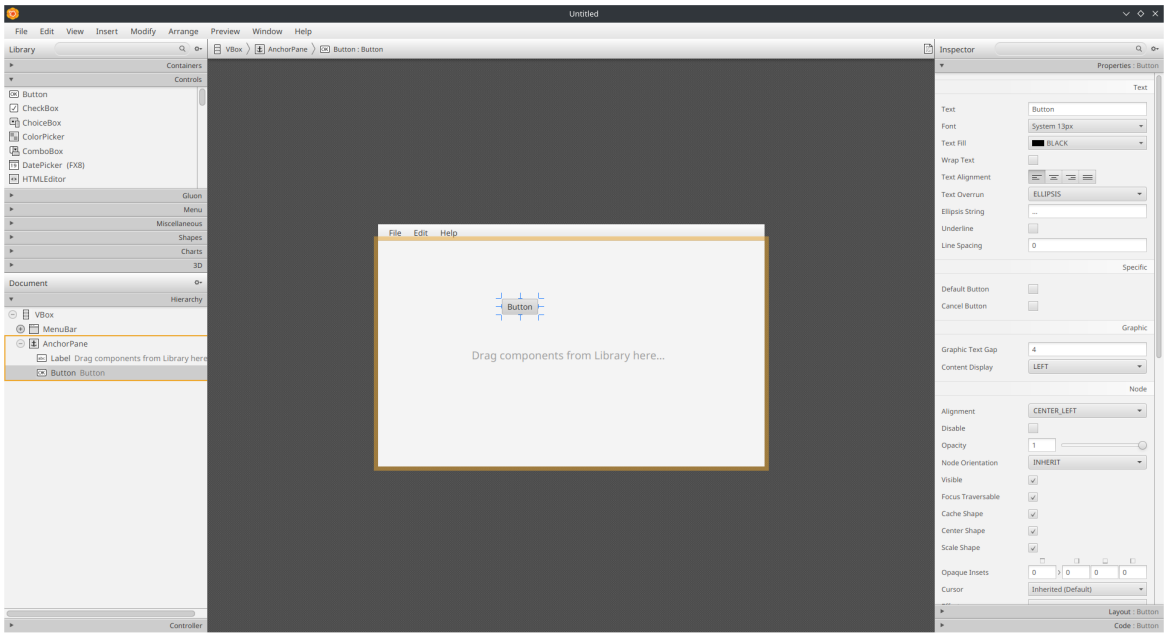
Scene Builder es una alternativa orientada al diseño que puede ser más productiva. Además es multiplataforma y está disponible para GNU/Linux, Windows y Mac. Scene Builder funciona con el ecosistema JavaFX: controles oficiales, proyectos comunitarios y ofertas de Gluon que incluyen Gluon Mobile, Gluon Desktop y Gluon CloudLink.

El diseño de la interfaz de usuario *drag&drop* permite una iteración rápida. La separación de los archivos de diseño y lógica permite que los miembros del equipo se concentren rápida y fácilmente en su capa específica de desarrollo de aplicaciones.

Scene Builder es gratuito y de código abierto, pero cuenta con el respaldo de Gluon. Están disponibles ofertas de soporte comercial, que incluyen formación y servicios de consultoría personalizados.

Descarga e información: https://gluonhq.com/products/scene-builder/




Untitled
File Edit View Insert Modify Arrange Preview Window Help
Library
Containers
Controls
Button
CheckBox
ChoiceBox
ColorPicker
ComboBox
DatePicker (FX8)
HTMLEditor
Gluon
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
VBox
MenuBar
AnchorPane
Label Drag components from Library here
Button Button
Controller
VBox AnchorPane Button : Button
File Edit Help
Button
Drag components from Library here...
Inspector
Properties : Button
Text
Text Button
Font System 13px
Text Fill BLACK
Wrap Text
Text Alignment
Text Overrun ELLIPSIS
Ellipsis String ...
Underline
Line Spacing 0
Specific
Default Button
Cancel Button
Graphic
Graphic Text Gap 4
Content Display LEFT
Node
Alignment CENTER_LEFT
Disable
Opacity 1
Node Orientation INHERIT
Visible
Focus Traversable
Cache Shape
Center Shape
Scale Shape
Opaque Insets 0 0 0 0
Cursor Inherited (Default)
Layout : Button
Code : Button

6. Mejores prácticas

6.1. Propiedades estilizables

Se puede diseñar una propiedad JavaFX a través de css usando `StyleableProperty`. Esto es útil cuando los controles necesitan propiedades que se pueden configurar a través de css.

Para usar `StyleableProperty` en un Control, se necesita crear un nuevo `CssMetaData` usando `StyleableProperty`. Los `CssMetaData` creados para un control deben agregarse a `List<CssMetaData>` obtenidos del antecesor del control. Esta nueva lista luego se devuelve desde el archivo `getControlCssMetaData()`.

Por convención, las clases de control que tienen `CssMetaData` implementarán un método estático `getClassCssMetaData()` y es habitual que `getControlCssMetaData()` simplemente devuelva `getClassCssMetaData()`. El propósito de `getClassCssMetaData()` es permitir que las subclases incluyan fácilmente los `CssMetaData` de algún antepasado.

```
1  // StyleableProperty
2  private final StyleableProperty<Color> color =
3      new SimpleStyleableObjectProperty<>(COLOR, this, "color");
4
5  // Typical JavaFX property implementation
6  public Color getColor() {
7      return this.color.getValue();
8  }
9  public void setColor(final Color color) {
10     this.color.setValue(color);
11 }
12 public ObjectProperty<Color> colorProperty() {
13     return (ObjectProperty<Color>) this.color;
14 }
15
16 // CssMetaData
17 private static final CssMetaData<MY_CTRL, Paint> COLOR =
18     new CssMetaData<MY_CTRL, Paint>("-color", PaintConverter.getInstance(), Color.RED)
   {
19
20     @Override
21     public boolean isSettable(MY_CTRL node) {
22         return node.color == null || !node.color.isBound();
23     }
24
25     @Override
26     public StyleableProperty<Paint> getStyleableProperty(MY_CTRL node) {
27         return node.color;
28     }
29 };
30
31 private static final List<CssMetaData<? extends Styleable, ?>> STYLEABLES;
```

```
static {
    // Fetch CssMetaData from its ancestors
    final List<CssMetaData<? extends Styleable, ?>> styleables =
        new ArrayList<>(Control.getClassCssMetaData());
    // Add new CssMetaData
    styleables.add(COLOR);
    STYLEABLES = Collections.unmodifiableList(styleables);
}

// Return all CssMetadata information
public static List<CssMetaData<? extends Styleable, ?>> getClassCssMetaData() {
    return STYLEABLES;
}

@Override
public List<CssMetaData<? extends Styleable, ?>> getControlCssMetaData() {
    return getClassCssMetaData();
}
```

La creación de `StyleableProperty` y `CssMetaData` necesita una gran cantidad de código repetitivo y esto se puede reducir mediante el uso de `StyleablePropertyFactory` . `StyleablePropertyFactory` contiene métodos para crear `StyleableProperty` con los `CssMetaData` correspondientes.

```
// StyleableProperty
private final StyleableProperty<Color> color =
    new SimpleStyleableObjectProperty<>(COLOR, this, "color");

// Typical JavaFX property implementation
public Color getColor() {
    return this.color.getValue();
}
public void setColor(final Color color) {
    this.color.setValue(color);
}
public ObjectProperty<Color> colorProperty() {
    return (ObjectProperty<Color>) this.color;
}

// StyleablePropertyFactory
private static final StyleablePropertyFactory<MY_CTRL> FACTORY =
    new StyleablePropertyFactory<>(Control.getClassCssMetaData());

// CssMetaData from StyleablePropertyFactory
private static final CssMetaData<MY_CTRL, Color> COLOR =
    FACTORY.createColorCssMetaData("-color", s -> s.color, Color.RED, false);

// Return all CssMetadata information from StyleablePropertyFactory
public static List<CssMetaData<? extends Styleable, ?>> getClassCssMetaData() {
    return FACTORY.getCssMetaData();
}

```

```
29  @Override public List<CssMetaData<? extends Styleable, ?>> getControlCssMetaData() {
30      return getClassCssMetaData();
31  }
```

6.2. Tareas

Ahora veremos cómo usar una tarea JavaFX para mantener la IU *responsible*. Es imperativo que cualquier operación que tarde más de unos pocos cientos de milisegundos se ejecute en un subproceso separado para evitar bloquear la interfaz de usuario. Una tarea concluye la secuencia de pasos en una operación de larga duración y proporciona devoluciones de llamada para los posibles resultados.

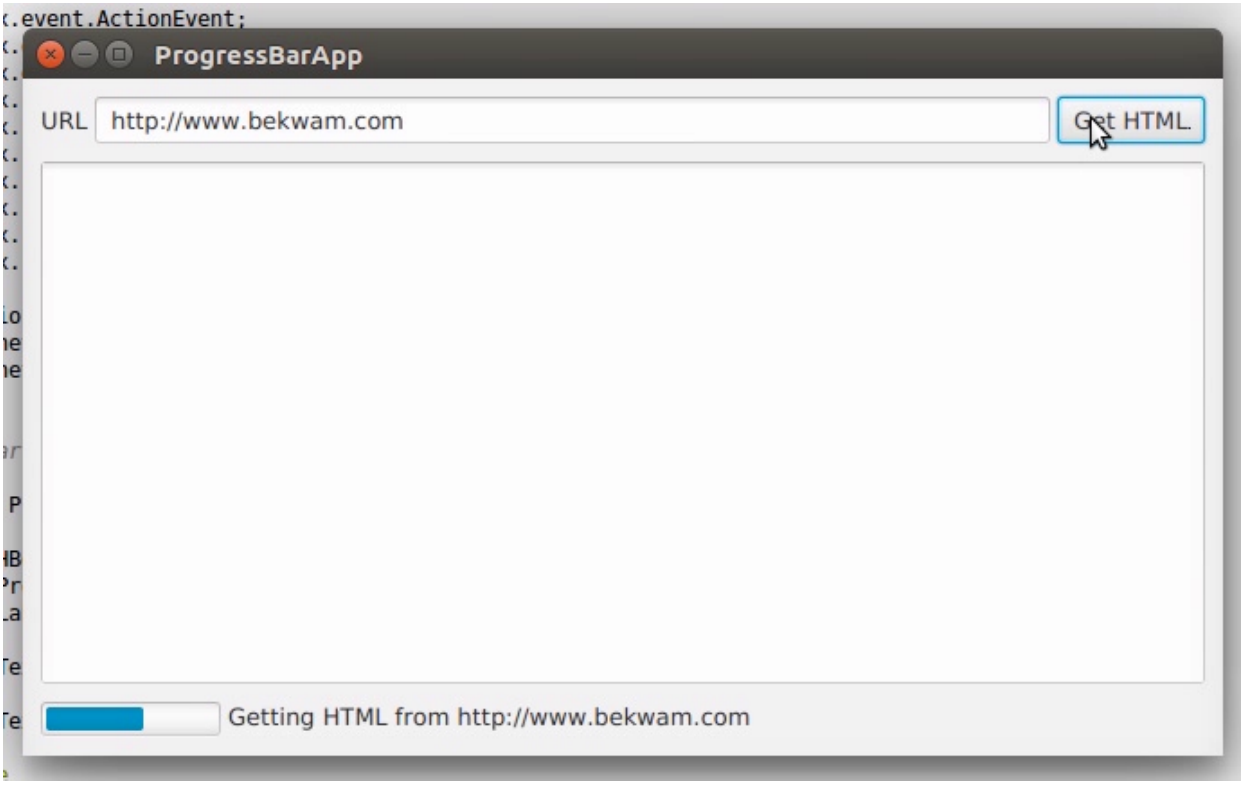
La clase **Task** también mantiene al usuario al tanto de la operación a través de propiedades que se pueden vincular a controles de interfaz de usuario como `ProgressBars` y `Labels`. El enlace actualiza dinámicamente la interfaz de usuario. Estas propiedades incluyen

1. `runningProperty` : si la tarea se está ejecutando o no
2. `ProgressProperty` : el porcentaje completado de una operación.
3. `messageProperty` : texto que describe un paso en la operación

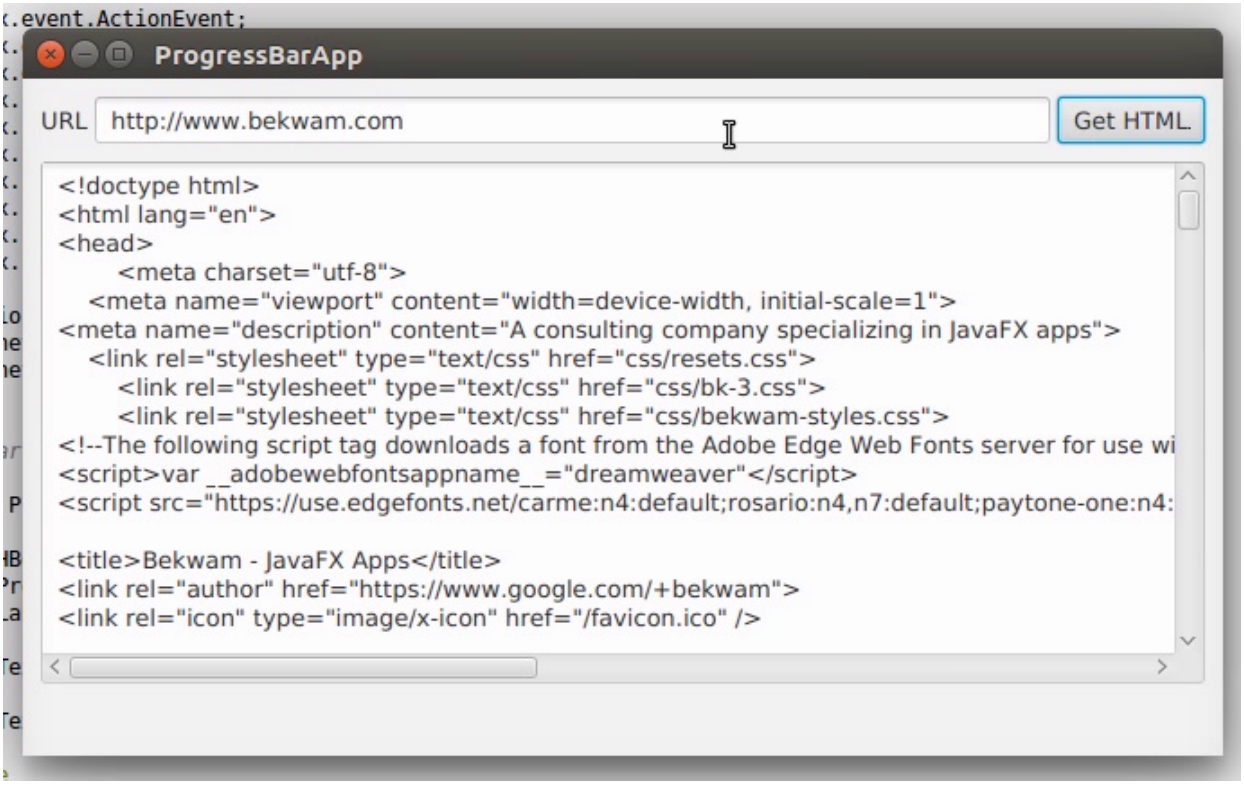
6.2.1. Demostración

Las siguientes capturas de pantalla muestran el funcionamiento de una aplicación de recuperación de HTML.

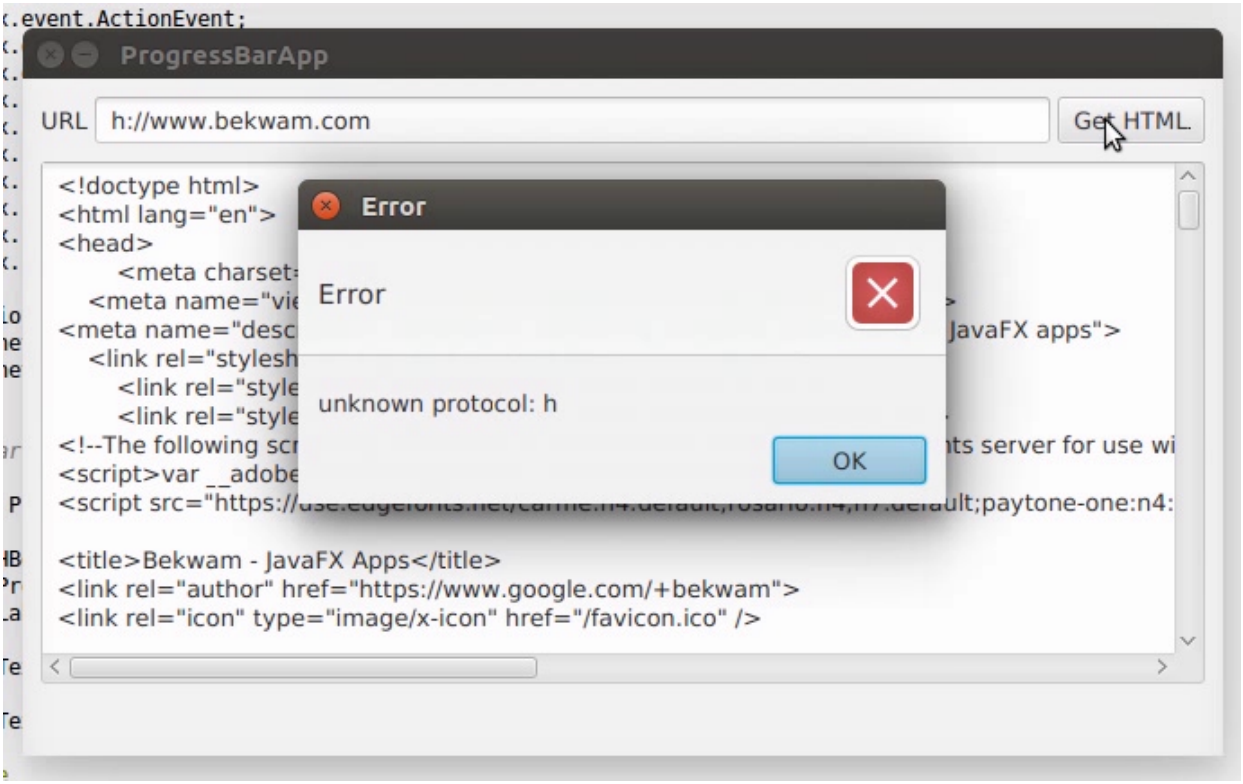
Ingresar una URL y presionar "Ir" iniciará una tarea JavaFX. Al ejecutarse, la tarea hará visible un `HBox` que contiene una barra de progreso y una etiqueta. `ProgressBar` y `Label` se actualizan a lo largo de la operación.



Cuando finaliza la recuperación, se invoca al metodo `succeeded()` y se actualiza la interfaz de usuario. Tenga en cuenta que la llamada a `succeeded()` se lleva a cabo en el subproceso FX, por lo que es seguro manipular los controles.



Si hubo un error al recuperar el HTML, se invoca a `failed()` y se muestra una alerta de error. `failed()` también tiene lugar en el subproceso FX. Esta captura de pantalla muestra una entrada no válida. Se usa una "h" en la URL en lugar de "http".



6.2.2. Código

Se coloca un controlador de eventos en el botón Obtener HTML que crea la tarea. El punto de entrada de la Tarea es el método call() que comienza llamando a updateMessage() y updateProgress(). Estos métodos se ejecutan en el subproceso FX y generarán actualizaciones en cualquier propiedad enlazada.

El programa continúa emitiendo un HTTP GET usando clases estándar de java.net. Se crea una cadena "retval" a partir de los caracteres recuperados. Las propiedades de mensaje y progreso se actualizan con más llamadas a updateMessage() y updateProgress(). El método call() finaliza con la devolución de la cadena que contiene el texto HTML.

En una operación exitosa, se invoca la devolución de llamada de éxito (). getValue() es un método de tarea que devolverá el valor acumulado en la tarea (recuerde "retval"). El tipo del valor es lo que se proporciona en el argumento genérico, en este caso "String". Esto podría ser un tipo complejo como un objeto de dominio o una colección. La operación de éxito () se ejecuta en el subproceso FX, por lo que la cadena getValue () se establece directamente en el área de texto.

Si la operación falla, se lanza una excepción. La excepción es capturada por la tarea y convertida en una llamada fallida(). fail() también es seguro para subprocesos FX y muestra una alerta.

```
String url = tfURL.getText();

Task<String> task = new Task<String>() {

    @Override
    protected String call() throws Exception {

        updateMessage("Getting HTML from " + url );
        updateProgress( 0.5d, 1.0d );

        HttpURLConnection c = null;
        InputStream is = null;
        String retval = "";

        try {

            c = (HttpURLConnection) new URL(url).openConnection();

            updateProgress( 0.6d, 1.0d );
            is = c.getInputStream();
            int ch;
            while( (ch=is.read()) != -1 ) {
                retval += (char)ch;
            }

        } finally {
            if( is != null ) {
                is.close();
            }
            if( c != null ) {
```

```
                c.disconnect();
            }
        }

        updateMessage("HTML retrieved");
        updateProgress( 1.0d, 1.0d );

        return retval;
    }

    @Override
    protected void succeeded() {
        contents.setText( getValue() );
    }

    @Override
    protected void failed() {
        Alert alert = new Alert(Alert.AlertType.ERROR, getException().getMessage() );
        alert.showAndWait();
    }
};
```

Tenga en cuenta que la tarea no actualiza la barra de progreso y la etiqueta de estado directamente. En su lugar, Task realiza llamadas seguras a updateMessage() y updateProgress(). Para actualizar la interfaz de usuario, se utiliza el enlace JavaFX en las siguientes declaraciones.

```
bottomControls.visibleProperty().bind( task.runningProperty() );
pb.progressProperty().bind( task.progressProperty() );
messageLabel.textProperty().bind( task.messageProperty() );
```

Task.runningProperty es un valor booleano que se puede vincular a bottomControls HBox visibleProperty. Task.progressProperty es un doble que se puede vincular a ProgressBarprogressProperty. Task.messageProperty es una cadena que se puede vincular a la etiqueta de estado textProperty.

Para ejecutar la tarea, cree un subproceso que proporcione la tarea como argumento del constructor e invoque start().

```
new Thread(task).start();
```

Para cualquier operación de ejecución prolongada (archivo IO, la red), use una tarea JavaFX para mantener la capacidad de respuesta de su aplicación. La tarea JavaFX le brinda a su aplicación una forma consistente de manejar operaciones asincrónicas y expone varias propiedades que se pueden usar para eliminar la lógica repetitiva y de programación.

6.2.3. Código completo

El código se puede probar en un solo archivo .java.

```
public class ProgressBarApp extends Application {

    private HBox bottomControls;
    private ProgressBar pb;
    private Label messageLabel;

    private TextField tfURL;

    private TextArea contents;

    @Override
    public void start(Stage primaryStage) throws Exception {

        Parent p = createMainView();

        Scene scene = new Scene(p);

        primaryStage.setTitle("ProgressBarApp");
        primaryStage.setWidth( 667 );
        primaryStage.setHeight( 376 );
        primaryStage.setScene( scene );
        primaryStage.show();
    }

    private Parent createMainView() {

        VBox vbox = new VBox();
        vbox.setPadding( new Insets(10) );
        vbox.setSpacing( 10 );

        HBox topControls = new HBox();
        topControls.setAlignment(Pos.CENTER_LEFT);
        topControls.setSpacing( 4 );

        Label label = new Label("URL");
        tfURL = new TextField();
        HBox.setHgrow( tfURL, Priority.ALWAYS );
        Button btnGetHTML = new Button("Get HTML");
        btnGetHTML.setOnAction( this::getHTML );
        topControls.getChildren().addAll(label, tfURL, btnGetHTML);

        contents = new TextArea();
        VBox.setVgrow( contents, Priority.ALWAYS );

        bottomControls = new HBox();
        bottomControls.setVisible(false);
        bottomControls.setSpacing( 4 );
```

```
48          HBox.setMargin( bottomControls, new Insets(4));
49
50          pb = new ProgressBar();
51          messageLabel = new Label("");
52          bottomControls.getChildren().addAll(pb, messageLabel);
53
54          vbox.getChildren().addAll(topControls, contents, bottomControls);
55
56          return vbox;
57      }
58
59      public void getHTML(ActionEvent evt) {
60
61          String url = tfURL.getText();
62
63          Task<String> task = new Task<String>() {
64
65              @Override
66              protected String call() throws Exception {
67
68                  updateMessage("Getting HTML from " + url );
69                  updateProgress( 0.5d, 1.0d );
70
71                  HttpURLConnection c = null;
72                  InputStream is = null;
73                  String retval = "";
74
75                  try {
76
77                      c = (HttpURLConnection) new URL(url).openConnection();
78
79                      updateProgress( 0.6d, 1.0d );
80                      is = c.getInputStream();
81                      int ch;
82                      while( (ch=is.read()) != -1 ) {
83                          retval += (char)ch;
84                      }
85
86                  } finally {
87                      if( is != null ) {
88                          is.close();
89                      }
90                      if( c != null ) {
91                          c.disconnect();
92                      }
93                  }
94
95                  updateMessage("HTML retrieved");
96                  updateProgress( 1.0d, 1.0d );
97
98                  return retval;
99              }
```

```
100
101             @Override
102             protected void succeeded() {
103                 contents.setText( getValue() );
104             }
105
106             @Override
107             protected void failed() {
108                 Alert alert = new Alert(Alert.AlertType.ERROR,
getException().getMessage() );
109                 alert.showAndWait();
110             }
111         };
112
113         bottomControls.visibleProperty().bind( task.runningProperty() );
114         pb.progressProperty().bind( task.progressProperty() );
115         messageLabel.textProperty().bind( task.messageProperty() );
116
117         new Thread(task).start();
118     }
119
120     public static void main(String[] args) {
121         launch(args);
122     }
123 }
```

6.3. Evitar Nulos en ComboBoxes

Para usar a `ComboBox` en JavaFX, declare una Lista de elementos y establezca un valor inicial usando setValue(). El `ComboBox` método getValue() recupera el valor seleccionado actualmente. Si no se proporciona un valor inicial, el control tiene un valor nulo predeterminado.

El valor nulo es un problema cuando `ComboBox` impulsa otra lógica como una transformación a mayúsculas o la búsqueda de un registro de base de datos. Si bien generalmente se usa una verificación nula para evitar este tipo de error, se prefiere un objeto vacío para simplificar el código. Los cuadros combinados a menudo aparecen en grupos y la técnica de objetos vacíos reduce las comprobaciones nulas en la interacción de los cuadros combinados relacionados y en las operaciones de guardar y cargar.

Este artículo presenta un par de ComboBoxes relacionados. Una selección de país en uno `ComboBox` modifica la lista de elementos de ciudad disponibles en un segundo `ComboBox`. No se requiere ninguna selección. El usuario puede presionar Guardar `Button` en cualquier momento y, si no se realiza ninguna selección `ComboBox`, se devolverá un objeto vacío, en este caso una Cadena vacía.

Esta es una captura de pantalla de la aplicación. Si selecciona "Suiza" de un valor inicial vacío, la ciudad se llenará `ComboBox` de ciudades suizas. Seleccionando la ciudad "Zurich" y presionando Guardar recuperará esos valores.

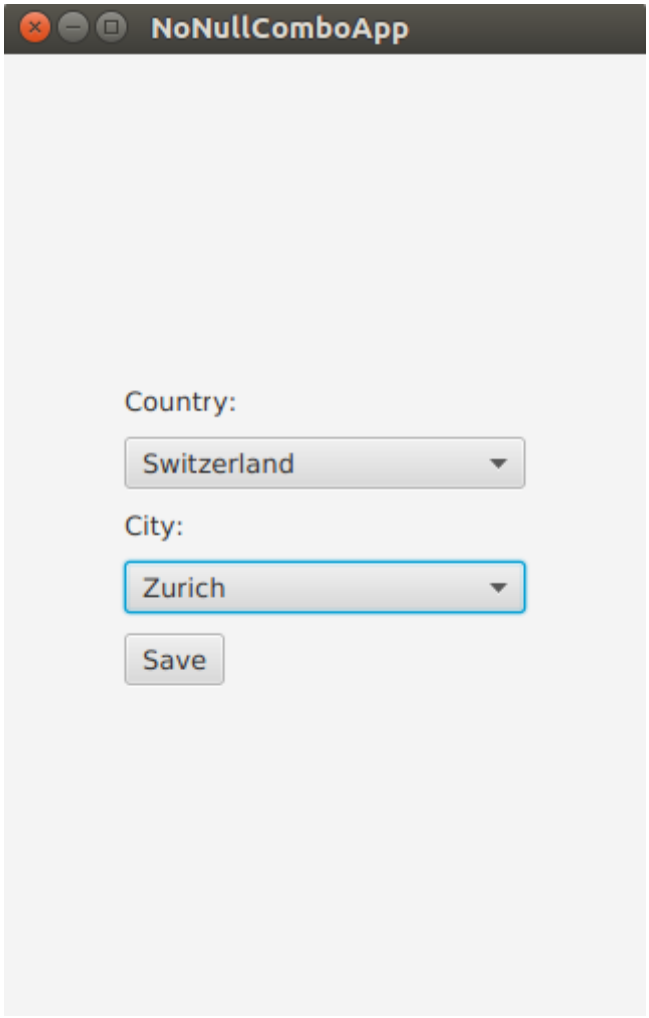


Figura 64. Cuadros combinados relacionados

6.3.1. Estructura de datos

Las estructuras de datos que soportan la aplicación son una Lista de países y un Mapa de ciudades. El Mapa de ciudades utiliza el país como clave.

NoNullComboApp.clase

```
1  public class NoNullComboApp extends Application {
2
3      private List<String> countries = new ArrayList<>();
4
5      private Map<String, List<String>> citiesMap = new LinkedHashMap<>();
6
7      private void initData() {
8
9          String COUNTRY_FR = "France";
10         String COUNTRY_DE = "Germany";
11         String COUNTRY_CH = "Switzerland";
12
13         countries.add(COUNTRY_FR); countries.add(COUNTRY_DE);
   countries.add(COUNTRY_CH);
14
```

```
        List<String> frenchCities = new ArrayList<>();
        frenchCities.add("Paris");
        frenchCities.add("Strasbourg");

        List<String> germanCities = new ArrayList<>();
        germanCities.add("Berlin");
        germanCities.add("Cologne");
        germanCities.add("Munich");

        List<String> swissCities = new ArrayList<>();
        swissCities.add("Zurich");

        citiesMap.put(COUNTRY_FR, frenchCities );
        citiesMap.put(COUNTRY_DE, germanCities );
        citiesMap.put(COUNTRY_CH, swissCities );
    }
```

Para recuperar el conjunto de ciudades de un país determinado, utilice el método get() del Mapa. El método containsKey() se puede utilizar para determinar si el mapa contiene o no un valor para el país especificado. En este ejemplo, containsKey() se usará para manejar el caso del objeto vacío.

6.3.2. interfaz de usuario

La interfaz de usuario es un par de cuadros combinados con etiquetas y un botón Guardar. Los controles se colocan en a `VBox` y justificados a la izquierda. El `VBox` está envuelto en un `TilePane` y centrado. Se `TilePane` utilizó ya que no se estira `VBox` horizontalmente.

NoNullComboApp.clase

```
    @Override
    public void start(Stage primaryStage) throws Exception {

        Label countryLabel = new Label("Country:");
        country.setPrefWidth(200.0d);
        Label cityLabel = new Label("City:");
        city.setPrefWidth(200.0d);
        Button saveButton = new Button("Save");

        VBox vbox = new VBox(
                countryLabel,
                country,
                cityLabel,
                city,
                saveButton
        );
        vbox.setAlignment(Pos.CENTER_LEFT );
        vbox.setSpacing( 10.0d );

        TilePane outerBox = new TilePane(vbox);
        outerBox.setAlignment(Pos.CENTER);

```

```
        Scene scene = new Scene(outerBox);

        initData();
```

6.3.3. Valores iniciales

Como se mencionó anteriormente, si no se especifica un valor para un `ComboBox`, se devolverá un valor nulo en una llamada a getValue(). Aunque existen varias técnicas defensivas (si se verifica, métodos Commons StringUtils) para defenderse de NullPointerExceptions, es mejor evitarlas por completo. Esto es especialmente cierto cuando las interacciones se vuelven complejas o hay varios ComboBoxes que permiten selecciones vacías.

NoNullComboApp.clase

```
        country.getItems().add("");
        country.getItems().addAll( countries );
        country.setValue( "" );  // empty selection is object and not null

        city.getItems().add("");
        city.setValue( "" );
```

En esta aplicación, el país `ComboBox` no se cambiará, por lo que sus elementos se agregan en el método start(). El país comienza con una selección inicial vacía al igual que la ciudad. Ciudad, en este punto, contiene un único elemento vacío.

6.3.4. Interacción

Cuando se cambia el valor del país, se `ComboBox` debe reemplazar el contenido de la ciudad. Es común usar clear() en la lista de respaldo; sin embargo, esto producirá un valor nulo en `ComboBox` (sin elementos, sin valor). En su lugar, use removeIf() con una cláusula para mantener un único elemento vacío. Con la lista limpia de todos los datos (excepto el elemento vacío), los contenidos recién seleccionados se pueden agregar con addAll().

NoNullComboApp.clase

```
        country.setOnAction( (evt) -> {

            String cty = country.getValue();

            city.getItems().removeIf( (c) -> !c.isEmpty() );

            if( citiesMap.containsKey(cty) ) {  // not an empty key
                city.getItems().addAll( citiesMap.get(cty) );
            }
        });

        saveButton.setOnAction( (evt) -> {
           System.out.println("saving country='" + country.getValue() +
                                    "', city='" + city.getValue() + "'");
        });
```

La acción del botón Guardar imprimirá los valores. En ningún caso se devolverá un valor nulo desde getValue().

Si es un desarrollador de Java, ha escrito "si no es nulo" miles de veces. Sin embargo, proyecto tras proyecto, veo NullPointerExceptions que resaltan los casos que se perdieron o las nuevas condiciones que surgieron. Este artículo presentó una técnica para mantener objetos vacíos en ComboBoxes estableciendo un valor inicial y usando removeIf() en lugar de clear() al cambiar listas. Aunque este ejemplo usó objetos String, esto se puede expandir para trabajar con objetos de dominio que tienen una implementación hashCode/equals, una representación de objeto vacía y cellFactory o toString() para producir una vista vacía.

6.3.5. Código completo

El código se puede probar en un solo archivo .java.

NoNullComboApp.clase

```
public class NoNullComboApp extends Application {

    private final ComboBox<String> country = new ComboBox<>();
    private final ComboBox<String> city = new ComboBox<>();

    private List<String> countries = new ArrayList<>();

    private Map<String, List<String>> citiesMap = new LinkedHashMap<>();

    @Override
    public void start(Stage primaryStage) throws Exception {

        Label countryLabel = new Label("Country:");
        country.setPrefWidth(200.0d);
        Label cityLabel = new Label("City:");
        city.setPrefWidth(200.0d);
        Button saveButton = new Button("Save");

        VBox vbox = new VBox(
                countryLabel,
                country,
                cityLabel,
                city,
                saveButton
        );
        vbox.setAlignment(Pos.CENTER_LEFT );
        vbox.setSpacing( 10.0d );

        TilePane outerBox = new TilePane(vbox);
        outerBox.setAlignment(Pos.CENTER);

        Scene scene = new Scene(outerBox);

        initData();
```

```
35
36          country.getItems().add("");
37          country.getItems().addAll( countries );
38          country.setValue( "" );  // empty selection is object and not null
39
40          city.getItems().add("");
41          city.setValue( "" );
42
43          country.setOnAction( (evt) -> {
44
45              String cty = country.getValue();
46
47              city.getItems().removeIf( (c) -> !c.isEmpty() );
48
49              if( citiesMap.containsKey(cty) ) {  // not an empty key
50                  city.getItems().addAll( citiesMap.get(cty) );
51              }
52          });
53
54          saveButton.setOnAction( (evt) -> {
55             System.out.println("saving country='" + country.getValue() +
56                                       "', city='" + city.getValue() + "'");
57          });
58
59          primaryStage.setTitle("NoNullComboApp");
60          primaryStage.setScene( scene );
61          primaryStage.setWidth( 320 );
62          primaryStage.setHeight( 480 );
63          primaryStage.show();
64      }
65
66      public static void main(String[] args) {
67          launch(args);
68      }
69
70      private void initData() {
71
72          String COUNTRY_FR = "France";
73          String COUNTRY_DE = "Germany";
74          String COUNTRY_CH = "Switzerland";
75
76          countries.add(COUNTRY_FR); countries.add(COUNTRY_DE);
    countries.add(COUNTRY_CH);
77
78          List<String> frenchCities = new ArrayList<>();
79          frenchCities.add("Paris");
80          frenchCities.add("Strasbourg");
81
82          List<String> germanCities = new ArrayList<>();
83          germanCities.add("Berlin");
84          germanCities.add("Cologne");
85          germanCities.add("Munich");
```

```
86
87          List<String> swissCities = new ArrayList<>();
88          swissCities.add("Zurich");
89
90          citiesMap.put(COUNTRY_FR, frenchCities );
91          citiesMap.put(COUNTRY_DE, germanCities );
92          citiesMap.put(COUNTRY_CH, swissCities );
93      }
94  }
```

7. Píldoras informáticas relacionadas

- https://www.youtube.com/playlist?list=PLNjWMbvTJAIjLRW2qyuc4DEgFVW5YFRSR
- https://www.youtube.com/playlist?list=PLaxZkGlLWHGUWZxuadN3J7KKaICRlhz5-

8. Fuentes de información

- Wikipedia
- Programación (Grado Superior) - Juan Carlos Moreno Pérez (Ed. Ra-ma)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- Apuntes José Luis Comesaña
- Apuntes IOC Programació bàsica (Joan Arnedo Moreno)
- Apuntes IOC Programació Orientada a Objectes (Joan Arnedo Moreno)
- FXDocs
- https://openjfx.io/openjfx-docs/